

Long-term software reliability prediction via nonparametric kernel regression methods

Jiacheng Bai^{*,}, Tadashi Dohi, Jingchi Wu[,], Junjun Zheng[,], Hiroyuki Okamura[,]

Graduate School of Advanced Science and Engineering, Hiroshima University, 1-4-1 Kagamiyama, Higashi-Hiroshima, Hiroshima, 739-8527, Japan

ARTICLE INFO

Dataset link: [Software Fault-Detection Data \(Original data\)](#)

Keywords:

Software reliability
Nonparametric prediction
Kernel regression
Multi-kernel learning
Data-driven approach

ABSTRACT

Context: Software reliability prediction is essential for quality assurance and release decisions in modern software development. Long-horizon prediction remains challenging, especially when fault-count data are limited and testing conditions evolve over time.

Objective: This paper aims to develop a unified nonparametric kernel regression approach for long-horizon software reliability prediction, and to clarify the effects of structural flexibility and the sequential updating strategy on predictive performance.

Methods: We propose a unified nonparametric kernel regression framework that integrates both single-kernel and multi-kernel estimators with a closed-loop recursive forecasting protocol. The multi-kernel extension represents software reliability growth as a combination of heterogeneous kernels, where the kernel weights and bandwidths are simultaneously optimized by least-squares cross-validation. Experiments are conducted on sixteen real-world software development project datasets, including both classical closed-source and open-source projects on GitHub, under three training cutoffs (20%, 50%, and 80%), corresponding to the early, middle, and late testing phases.

Results: The results show that the proposed kernel-based models achieve competitive point-prediction performance compared with representative NHPP-based, wavelet-based, and LSTM-based baselines across the three testing phases. In the early testing phase, MKLL-D achieves the largest number of best-performing cases, while LSTM and Wavelet also perform best on some datasets. In the middle testing phase, Wavelet shows strong dataset-wise consistency, whereas LL-based multi-kernel models remain competitive. In the late testing phase, LSTM and NHPP-based SRMs become more competitive, while LL-based kernel models remain competitive on several datasets.

Conclusion: The proposed framework provides a practical nonparametric approach to long-horizon software reliability prediction. The results suggest that multi-kernel local linear models are especially useful in the early testing phase, as shown by the strong performance of MKLL-D. However, this advantage is not consistent across all datasets or phases. Sequential updating is mainly helpful for early-stage local linear estimators; in the middle phase, fixed-parameter configurations become more competitive, and in the late phase the relative performance becomes more dataset-dependent.

1. Introduction

Software reliability models (SRMs) play a fundamental role in modern software engineering by providing quantitative measures for assessing system reliability and supporting testing management throughout the software development lifecycle [1]. During software testing, software failures are typically observed as fault counts by a stochastic process over time, representing the cumulative number of detected faults. Modeling such fault-count processes enables the estimation of key reliability metrics, including the probability that a software system

operates without failure over a specified future interval. These metrics are useful for release decision-making and quality assurance.

Despite its importance, early-stage reliability prediction remains a significant challenge. In contemporary software projects, reliability prediction is increasingly required at the early or middle testing phases, when only limited fault data are available and uncertainty is inherently high. This challenge becomes further complicated by the growing scale and complexity of modern software systems, whose testing efforts, debugging efficiency, and operational conditions evolve dynamically.

* Corresponding author.

E-mail address: d252433@hiroshima-u.ac.jp (J. Bai).

<https://doi.org/10.1016/j.infsof.2026.108284>

Received 17 April 2026; Received in revised form 24 June 2026; Accepted 28 July 2026

Available online 1 August 2026

0950-5849/© 2026 Elsevier B.V. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

Under such non-stationary testing environments, accurately characterizing the evolving fault-detection process is regarded as a central and nontrivial challenge for software reliability modeling.

To address this problem, a wide range of SRMs have been proposed, among which parametric models based on nonhomogeneous Poisson processes (NHPPs) remain the dominant paradigm. Representative examples include the Goel–Okumoto exponential SRM [2] and Yamada et al.’s delayed S-shaped SRM [3], which characterize reliability growth patterns through prespecified mean value functions. Owing to their analytical tractability and interpretability, NHPP-based SRMs have been widely adopted in practice. However, the structural rigidity of these models, which stems from prespecified mean value functions, may lead to extrapolation bias when fault-detection dynamics deviate from the assumed patterns. In real software testing processes, fault-detection dynamics are influenced by multiple time-varying factors, such as fluctuating testing effort, uneven debugging complexity, and requirement changes [1,4]. As a result, a single parametric SRM may not remain adequate throughout the entire testing lifecycle.

Adaptive extensions, such as Kalman-filter-based formulations, have been investigated to improve the adaptability and tracking capability of software reliability models [5]. These approaches usually track the evolution of software reliability by updating latent reliability states based on observed failure data. However, in long-horizon prediction, future fault counts are unavailable at the time of forecasting. Therefore, the recursive procedure in this study is not an update based on newly observed failures, but a closed-loop forecasting protocol that uses previously predicted cumulative fault counts to construct an expanding prediction trajectory. In addition, parametric SRMs remain constrained by their prespecified mean value functions, which may cause extrapolation bias when the actual fault-detection process deviates from the assumed pattern. These considerations motivate flexible nonparametric approaches that reduce reliance on strong structural assumptions.

In this paper, we formulate a unified nonparametric regression approach that reformulates software reliability prediction as a mean-value-based regression problem for grouped cumulative fault-count processes. Unlike conventional NHPP-based SRMs, the proposed framework does not impose a prespecified parametric form on the mean value function. Instead, it directly estimates the underlying mean value function by locally smoothing the observed grouped cumulative fault-count data. From this perspective, kernel-based regression provides a natural modeling paradigm because it offers local adaptivity, an interpretable local weighting mechanism, and data-adaptive smoothing capability under limited-data conditions.

Within the proposed framework, classical kernel-based estimators, such as the Nadaraya–Watson (NW) estimator [6,7] and the local linear estimator [8], are used as basic model configurations. We further formulate their multi-kernel counterparts, in which the smoothing kernel is represented as a convex combination of multiple base kernels. This construction allows different kernel shapes and smoothing scales to be incorporated into a unified estimator, so that global reliability trends and local variations can be represented within the same modeling framework.

A key practical challenge in kernel-based modeling lies in controlling model complexity. Inappropriate choices of bandwidths or kernel configurations may lead to excessive smoothing or overfitting, especially when fault-count data are sparse. To address this issue, we use least-squares cross-validation (LSCV) to select smoothing parameters in a data-driven manner. This tuning mechanism provides a systematic way to reduce manual parameter selection and to balance the bias–variance trade-off under different testing phases.

To provide a broader empirical comparison, we include AIC-selected traditional NHPP-based SRMs, wavelet-based prediction models, and LSTM-based prediction models as representative baselines. These baselines allow us to compare the proposed framework with representative parametric, signal-processing, and learning-based prediction approaches under the same long-horizon forecasting setting.

Building on existing kernel regression and multi-kernel learning techniques, this study adapts them to long-horizon software reliability prediction with grouped cumulative fault-count data. The focus is placed on a unified recursive forecasting framework, where single-kernel and multi-kernel estimators are compared across different testing phases in terms of point prediction performance.

The main contributions of this paper are summarized as follows:

- We formulate a unified mean-value-based nonparametric point-prediction framework for grouped cumulative fault-count data, in which single-kernel and multi-kernel smoothing estimators are formulated under a common forecasting setting.
- We use an LSCV-based objective to determine smoothing parameters and multi-kernel weights in a data-driven manner.
- We construct multi-kernel smoothing estimators that combine multiple kernel functions and bandwidths to represent different temporal scales.
- We conduct a unified long-horizon point-prediction comparison against representative NHPP-based, wavelet-based, and LSTM-based baselines under three testing phases, and derive stage-aware model-selection recommendations.

The remainder of this paper is organized as follows. Section 2 reviews the evolution of parametric and nonparametric SRMs. Section 3 presents the mathematical formulation of the unified kernel regression framework and its optimization strategy. Section 4 describes the experimental setup. Section 5 reports experimental results on multiple software fault-count datasets. Section 6 concludes the paper and outlines future research directions.

2. Related work

Over the past five decades, software reliability modeling has emerged as an increasingly important research area, with extensive studies focusing on the stochastic characterization of software fault-detection processes. Among various approaches, parametric SRMs based on NHPPs have been widely adopted because of their mathematical tractability and interpretability. The seminal Goel–Okumoto SRM [2] assumes an exponential mean value function, capturing a monotonically decreasing fault-detection rate. Yamada et al.’s delayed S-shaped SRM [3] introduced delayed fault-detection behavior to reflect learning effects during software testing. Numerous NHPP-based SRMs with different functional forms have been proposed to model diverse reliability-growth patterns [9,10].

Despite their popularity, parametric NHPP-based SRMs rely on pre-specified mean value functions. In practice, software testing environments are often non-stationary, involving time-varying testing effort, debugging efficiency, and system complexity [1,4]. Under such conditions, a single parametric structure may be insufficient to characterize fault-detection dynamics over the entire testing lifecycle. Bayesian extensions have been proposed to improve model adaptability. However, limitations imposed by fixed functional forms may still remain [11]. Consequently, parametric SRMs may exhibit degraded long-term prediction performance when fault-detection behavior deviates from the assumed model structures.

To reduce dependence on rigid parametric assumptions, nonparametric SRMs have been investigated as flexible alternatives. Early studies focused on relaxing structural constraints on the mean value functions of NHPP-based models. Miller and Sofer [12] proposed a piecewise linear estimator, enabling greater flexibility without specifying a global functional form. Saito and Dohi [13] extended this idea to grouped fault-count data, which are commonly encountered in practice. In addition, methods from survival analysis, such as the Aalen estimator, have been introduced to estimate cumulative fault-intensity functions without parametric assumptions [14]. Saito and Dohi [15] applied constrained nonparametric maximum likelihood estimation to

software reliability modeling. While these approaches improve estimation flexibility, they mainly emphasize goodness-of-fit to the observed fault-count data and offer limited discussion on prediction stability across different testing phases. In particular, phase-dependent non-stationarity and data aggregation can induce discontinuities or sensitivity to grouping choices, which may lead to unstable long-term forecasts when the testing regime shifts.

In addition to statistical nonparametric models, machine learning techniques have also been applied to software reliability prediction. Neural network-based models, including recurrent neural networks and long short-term memory (LSTM) networks [16], have demonstrated competitive predictive performance in several studies [17]. Nevertheless, deep learning approaches often demand large-scale training samples to achieve convergence, which is frequently unattainable in the data-sparse early phases of software testing. In contrast, kernel-based regression offers data efficiency and statistical interpretability, making it particularly suitable for the data-sparse early and middle phases of software testing. However, the large data requirements and tuning complexity of deep learning models can restrict their applicability in these phases, where fault-count data are sparse and interpretability is crucial for reliability assessment and decision-making.

Wavelet-based nonparametric approaches have been explored to address non-stationarity and noise in software fault-count data. By decomposing fault-count processes into multi-resolution components, wavelet methods can separate long-term reliability trends from short-term fluctuations. Xiao and Dohi [18] applied wavelet shrinkage techniques to NHPP-based models, demonstrating improved goodness-of-fit under noisy conditions. Wu et al. [19] further extended wavelet-based methods for long-horizon reliability prediction. From a modeling perspective, wavelet techniques are often used as preprocessing or signal decomposition components within broader reliability modeling pipelines. Their performance strongly depends on the choice of wavelet basis and decomposition level, which may not be directly linked to the underlying fault-detection mechanism.

Kernel-based approaches provide a more direct nonparametric framework for modeling fault-detection dynamics. Early studies employed kernel density estimation to model the distribution of fault-detection times [20]. Subsequent work incorporated kernel regression into NHPP-based frameworks to estimate intensity and mean value functions [21]. A comprehensive approach using several kernel functions was proposed by Dohi et al. [22] as an extension of NHPP-based SRMs. Their study primarily focused on goodness-of-fit to historical observations, whereas long-horizon prediction was not the main emphasis. Local linear regression has also been applied to software fault-count data and has shown improved adaptability to non-stationary reliability-growth patterns [23]. However, most of them treat kernel regression mainly as a curve-fitting or estimation tool, while the connection between grouped cumulative fault-count data and counting-process-based mean-value decomposition has received limited attention.

Despite these advances, most existing kernel-based SRMs rely on a single kernel function with fixed bandwidth parameters, which are often selected heuristically. This setting implicitly assumes a uniform smoothness level across the testing process. As a result, it becomes difficult to capture global reliability trends and localized variations simultaneously. Moreover, repeated manual tuning of kernel and bandwidth configurations tends to increase computational cost and may reduce prediction stability [24].

In summary, parametric, nonparametric, and machine learning-based SRMs have each shown effectiveness in specific scenarios. However, unified frameworks that jointly consider structural adaptability, prediction stability, interpretability, and data efficiency remain limited. Existing kernel-based studies have demonstrated the usefulness of regression-based smoothing for software fault-count trends, but most of them still rely on single-kernel configurations or focus mainly on fitting and estimation rather than long-horizon prediction for grouped cumulative fault-count data.

Motivated by these observations, this paper formulates a mean-value-based kernel regression framework for grouped cumulative fault-count prediction. Within this formulation, both single-kernel and multi-kernel estimators are constructed under a consistent prediction setting. In particular, the proposed multi-kernel formulation learns kernel weights and smoothing parameters simultaneously, allowing the model to better accommodate heterogeneous and time-varying fault-detection dynamics across different phases of software testing.

3. Model description

3.1. NHPP-based SRMs

Suppose that software testing starts at time $t = 0$. Let $\{N(t), t \geq 0\}$ denote the cumulative number of software faults detected by time t . The counting process $N(t)$ is characterized by a non-decreasing stochastic point process, representing the progressive accumulation of software faults detected over time, and follows an NHPP satisfying the following conditions;

- $N(0) = 0$,
- The increments of $N(t)$ are statistically independent,
- $\Pr\{N(t + \Delta t) - N(t) \geq 2\} = o(\Delta t)$,
- $\Pr\{N(t + \Delta t) - N(t) = 1\} = \lambda(t)\Delta t + o(\Delta t)$,

where $\lambda(t)$ is a time-varying fault intensity function that governs the instantaneous fault-detection rate, and $o(\Delta t)$ denotes a higher order term in Δt . Under these assumptions, the probability mass function of $N(t)$ becomes

$$\Pr\{N(t) = n \mid N(0) = 0\} = \frac{\Lambda(t)^n \exp\{-\Lambda(t)\}}{n!},$$

where

$$\Lambda(t) = \int_0^t \lambda(s) ds,$$

is the mean value function, representing the expected cumulative number of software faults detected by time t .

Based on prior studies, NHPP-based SRMs can be classified into two types: Type I and Type II. In Type I, the mean value function is assumed to be increasing and unbounded:

$$\lim_{t \rightarrow \infty} \Lambda(t; \theta) = \infty.$$

In contrast, Type II assumes a bounded mean value function such that

$$\lim_{t \rightarrow \infty} \Lambda(t; \theta) = \omega,$$

where ω represents the expected total number of software faults before system testing. Accordingly, the mean value function in Type II is commonly expressed as

$$\Lambda(t; \theta) = \omega F(t; \alpha),$$

where $F(t; \alpha)$ is a cumulative distribution function (c.d.f.) characterizing the fault-detection time, and $\theta = (\omega, \alpha)$ denotes the parameter vector. This parametric formulation provides a clear and tractable modeling framework, making NHPP-based SRMs easy to interpret and apply in practice. So, Type I modeling allows imperfect debugging to insert new software faults, but Type II modeling implicitly assumes a fixed number of residual faults before testing which cannot be known.

To establish a benchmark for the comparison of NHPP-based SRMs, we employ ten representative parametric NHPP-based SRMs as baseline models. A summary of the selected baseline models is provided in Table 1, where the model parameters θ are estimated via maximum likelihood estimation.

Table 1
Representative parametric NHPP-based SRMs.

Models	$\Lambda(t; \theta)$
Exponential dist. (exp) [2]	$\Lambda(t; \theta) = \omega F(t; \alpha)$ $F(t; \alpha) = 1 - e^{-bt}$
Gamma dist. (gamma) [3,25]	$\Lambda(t; \theta) = \omega F(t; \alpha)$ $F(t; \alpha) = \int_0^t \frac{c^b s^{b-1} e^{-cs}}{\Gamma(b)} ds$
Pareto dist. (pareto) [26]	$\Lambda(t; \theta) = \omega F(t; \alpha)$ $F(t; \alpha) = 1 - \left(\frac{b}{t+b}\right)^c$
Truncated normal dist. (tnorm) [27]	$\Lambda(t; \theta) = \omega \frac{F(t; \alpha) - F(0; \alpha)}{1 - F(0; \alpha)}$ $F(t; \alpha) = \frac{1}{\sqrt{2\pi}b} \int_{-\infty}^t e^{-\frac{(s-\alpha)^2}{2b^2}} ds$
Log-normal dist. (lnorm) [27,28]	$\Lambda(t; \theta) = \omega F(\ln t; \alpha)$ $F(t; \alpha) = \frac{1}{\sqrt{2\pi}b} \int_{-\infty}^{\ln t} e^{-\frac{(s-\alpha)^2}{2b^2}} ds$
Truncated logistic dist. (tlogist) [29]	$\Lambda(t; \theta) = \omega F(t; \alpha)$ $F(t; \alpha) = \frac{1-e^{-bt}}{1+ce^{-bt}}$
Log-logistic dist. (llogist) [30]	$\Lambda(t; \theta) = \omega F(\ln t; \alpha)$ $F(t; \alpha) = \frac{(\ln t)^c}{1+(\ln t)^c}$
Truncated extreme-value max dist. (txvmax) [31]	$\Lambda(t; \theta) = \omega \frac{F(t; \alpha) - F(0; \alpha)}{1 - F(0; \alpha)}$ $F(t; \alpha) = e^{-e^{-\frac{t-\alpha}{b}}}$
Log-extreme-value max dist. (lxvmax) [31]	$\Lambda(t; \theta) = \omega F(\ln t; \alpha)$ $F(t; \alpha) = e^{-\left(\frac{t-\alpha}{b}\right)^c}$
Log-extreme-value min dist. (lxvmin) [2]	$\Lambda(t; \theta) = \omega (1 - F(-\ln t; \alpha))$ $F(t; \alpha) = e^{-e^{-\frac{t-\alpha}{b}}}$
$(\omega > 0, a > 0, b > 0, c > 0)$	

3.2. Kernel-based nonparametric formulation

Next, we formulate the proposed framework from a counting-process-based mean-value perspective for grouped cumulative fault-count data. Let $\{(t_i, N(t_i))\}_{i=1}^n$ denote the grouped observations, where t_i is the observation time and $N(t_i)$ is the cumulative number of detected faults up to t_i . Since grouped data provide only aggregated fault-count information at discrete observation times, the exact failure times within each interval are unavailable. Therefore, instead of specifying a complete stochastic law for individual failure times, we focus on estimating the mean value function that characterizes the cumulative fault-detection trend.

This viewpoint is motivated by the martingale-based formulation of counting processes, in which a counting process can be decomposed into a predictable cumulative intensity component and a martingale fluctuation component [32,33]. In particular, for a Poisson-type counting process with mean value function $\Lambda(t)$, the compensated process can be written as

$$M(t) = N(t) - \Lambda(t). \quad (1)$$

Under this formulation, $M(t)$ is a martingale with respect to the fault-detection history. Let $\mathcal{F}_s$ denote the information available up to time s . Then, for $s < t$, the martingale property implies

$$E[M(t) - M(s) | \mathcal{F}_s] = 0. \quad (2)$$

This means that, after the past fault-detection history and the predictable mean trend have been taken into account, the residual component has no predictable systematic drift. In other words, $M(t)$ represents the random fluctuation remaining after the cumulative fault-detection trend has been removed.

Accordingly, the cumulative fault-count process can be represented as

$$N(t) = \Lambda(t) + M(t), \quad (3)$$

where $\Lambda(t)$ represents the mean cumulative fault-detection trend and $M(t)$ represents the martingale residual component. At grouped observation times, this representation becomes

$$N(t_i) = \Lambda(t_i) + M(t_i), \quad i = 1, \dots, n. \quad (4)$$

It should be noted that this expression is not introduced as an ordinary additive regression assumption, nor does it require the exact failure times within each observation interval to be available. Rather, it provides a mean-value-based representation of grouped cumulative fault-count data motivated by counting-process theory.

For kernel-based estimation, the time variable is treated as the covariate of the mean value function. In the following subsections, we use t_i to denote the time point used in kernel estimation, and $N(t_i)$ denotes the corresponding cumulative number of detected faults. Thus, the grouped observations used by the proposed kernel estimators are written as $\{(t_i, N(t_i))\}_{i=1}^n$. The objective of the proposed kernel-based estimators is to estimate the mean value function $\Lambda(t)$ from these grouped cumulative observations. In the numerical experiments, the observation times are normalized to the unit interval before model fitting so that bandwidth parameters are comparable across datasets with different observation lengths. Under this formulation, both single-kernel and multi-kernel estimators can be regarded as nonparametric estimators of the mean value function within a unified long-horizon prediction framework.

3.3. Single-kernel SRMs

In this framework, single-kernel estimators constitute the most fundamental class of nonparametric SRMs. These models employ a kernel function with a fixed smoothing scale, and serve as building blocks for more advanced formulations [24].

3.3.1. Nadaraya–Watson estimator

We first consider the Nadaraya–Watson (NW) estimator, which is based on a local constant approximation of the reliability growth trend, and can be given by

$$\hat{\Lambda}_{NW}(t_0) = \frac{\sum_{i=1}^n w_i N(t_i)}{\sum_{i=1}^n w_i}, \quad (5)$$

where

$$w_i = K\left(\frac{t_0 - t_i}{h}\right)$$

is defined using a kernel function $K(\cdot)$ and a bandwidth parameter h . Owing to its simple formulation and low computational cost, the NW estimator serves as an intuitive and efficient baseline for kernel-based regression. However, the local constant assumption which is inherent in the NW estimator often leads to substantial bias near domain boundaries. Such boundary effects are particularly problematic in software reliability prediction.

3.3.2. Local linear estimator

We next consider the local linear (LL) estimator, which is attractive for its desirable boundary behavior and robustness. It addresses limitations of the NW estimator, particularly near data boundaries, where bias tends to increase. The basic form of the LL estimator is given by

$$\hat{\Lambda}_{LL}(t_0) = \frac{\sum_{i=1}^n w_i N(t_i)}{\sum_{i=1}^n w_i}, \quad (6)$$

where

$$w_i = K\left(\frac{t_0 - t_i}{h}\right) (s_{n,2} - (t_0 - t_i)s_{n,1})$$

with

$$s_{n,l} = \sum_{i=1}^n K\left(\frac{t_0 - t_i}{h}\right) (t_0 - t_i)^l, \quad l = 1, 2.$$

Here, the function $K(\cdot)$ is a kernel function, h denotes the bandwidth, and t_0 is the point of interest.

Table 2
Representative kernel functions.

Kernel	Mathematical form
Gaussian (Ga)	$K_1(u) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u^2}{2}\right)$
Triangular (Tr)	$K_2(u) = (1 - u)I_{[-1,1]}(u)$
Biweight (Bi)	$K_3(u) = \frac{15}{16}(1 - u^2)^2 I_{[-1,1]}(u)$
Epanechnikov (Ep)	$K_4(u) = \frac{3}{4}\left(1 - \frac{1}{5}u^2\right) / \sqrt{5} I_{[-\sqrt{5}, \sqrt{5}]}(u)$

3.3.3. Representative kernel functions

Several commonly used kernel functions are considered as candidate components in the proposed single-kernel and multi-kernel frameworks. It is well known that these kernels differ in smoothness and tail behavior, enabling the model to capture heterogeneous trend shapes.

$$I_{[a,b]}(u) \equiv \begin{cases} 1 & \text{if } u \in [a, b], \\ 0 & \text{otherwise.} \end{cases}$$

3.4. Multi-kernel estimators

While single-kernel models provide a solid foundation for nonparametric estimation, their expressive capacity is limited by both the choice of kernel function and the bandwidth parameter h [24]. The kernel function determines the weighting profile assigned to neighboring observations, whereas the bandwidth controls the smoothing scale. Therefore, a single-kernel estimator is restricted to one specific weighting pattern and one global smoothing scale. This restriction may be insufficient for software fault-count data, where the underlying fault-detection process can exhibit both gradual reliability growth and local temporal variations. In addition, the bandwidth parameter leads to the classical bias–variance trade-off: a larger bandwidth captures global reliability trends but tends to oversmooth local fault dynamics, whereas a smaller bandwidth preserves local variations at the cost of increased sensitivity to noise. In practice, identifying a single kernel function and a single bandwidth that simultaneously balance these competing objectives over the entire testing period remains challenging.

To address this limitation, we introduce a multi-kernel modeling strategy based on heterogeneous kernel components. The intuition is that different kernel functions impose different weighting profiles on the observed fault-count data. For example, the Gaussian kernel assigns nonzero weights to all observations and produces a smooth long-tailed weighting profile. In contrast, compactly supported kernels, such as the triangular, Epanechnikov, and biweight kernels, assign zero weights to observations outside a local neighborhood. These compact kernels also differ in how their weights decrease within the neighborhood; for instance, the triangular kernel decreases linearly, whereas the Epanechnikov and biweight kernels use polynomial weighting profiles. Therefore, using only one kernel function may restrict the estimator to a single weighting pattern.

In contrast to conventional approaches that rely on a single kernel function and a single smoothing scale, the proposed estimator allows each base kernel to differ in both functional form and bandwidth. By combining kernel functions with different support properties, decay patterns, and scale parameters, the model can represent a richer class of local weighting structures for software fault-count prediction.

Specifically, we construct a *composite kernel* as a convex combination of J base kernels with different bandwidths. The resulting multi-scale kernel is defined as

$$K_{MK}(u) = \sum_{j=1}^J \alpha_j K_{j,h_j}(u), \quad (7)$$

where $\alpha_j \geq 0$, $\sum_{j=1}^J \alpha_j = 1$, and

$$K_{j,h_j}(u) = \frac{1}{h_j} K_j\left(\frac{u}{h_j}\right). \quad (8)$$

In Eq. (8), the function $K_j(\cdot)$ denotes the j th base kernel function, such as the Gaussian, triangular, biweight, or Epanechnikov kernel listed in Table 2, and $h_j > 0$ is its associated bandwidth. The weight vector $\alpha = [\alpha_1, \dots, \alpha_J]^T$ allows the model to combine multiple smoothing scales adaptively. By incorporating this composite kernel into the single-kernel estimation framework, we derive the multi-kernel Nadaraya–Watson (MKNW) and multi-kernel local linear (MKLL) estimators.

3.4.1. Multi-kernel NW estimator

Substituting the composite kernel $K_{MK}(\cdot)$ into the standard NW formulation yields the multi-kernel (MKNW) estimator:

$$\hat{A}_{MKNW}(t) = \sum_{i=1}^n W_i^{MKNW}(t) N(t_i), \quad (9)$$

where the effective weights are determined by the normalized composite kernel values:

$$W_i^{MKNW}(t) = \frac{K_{MK}(t - t_i)}{\sum_{\ell=1}^n K_{MK}(t - t_\ell)}. \quad (10)$$

This estimator generalizes a zeroth-order approximation. By combining kernels with different bandwidths, it provides a richer hypothesis space for approximating the average reliability growth trend.

3.4.2. Multi-kernel LL estimator

To improve estimation accuracy, particularly near boundaries, we incorporate the composite kernel into the LL framework. Similar to the MKNW estimator, the resulting MKLL estimator is given by

$$\hat{A}_{MKLL}(t) = \sum_{i=1}^n W_i^{MKLL}(t) N(t_i). \quad (11)$$

The weights are derived from the solution to a locally weighted least-squares problem using $K_{MK}(\cdot)$:

$$W_i^{MKLL}(t) = \frac{K_{MK}(t - t_i) \left[S_{n,2}^{MK}(t) - (t - t_i) S_{n,1}^{MK}(t) \right]}{S_{n,0}^{MK}(t) S_{n,2}^{MK}(t) - \left(S_{n,1}^{MK}(t) \right)^2}. \quad (12)$$

where $S_{n,l}^{MK}(t)$ denotes the l th order discrete moment of the composite kernel:

$$S_{n,l}^{MK}(t) = \sum_{\ell=1}^n K_{MK}(t - t_\ell) (t - t_\ell)^l, \quad l = 0, 1, 2. \quad (13)$$

Using the composite moments allows the MKLL estimator to leverage multi-scale information, providing a robust approach for capturing complex temporal variations in observed software fault-count trends.

3.5. LSCV-based parameter selection for kernel estimators

Bandwidth selection is central to kernel-based mean value estimation because it determines how much neighboring information is used when estimating the software fault-count trend. A large bandwidth may average out important local changes, whereas a small bandwidth may overreact to noise in the observed data. This issue becomes more complicated in the multi-kernel setting, where both the bandwidths and the kernel weights must be determined. Therefore, manually selected or heuristic parameters may lead to unstable prediction performance, especially when the software fault-detection process changes over time.

Intuitively, least-squares cross-validation (LSCV) selects smoothing parameters by evaluating how well the estimator predicts observations that are temporarily left out. In a leave-one-out procedure, one observation is removed, the estimator is constructed from the remaining observations, and the removed observation is then predicted. If a set of bandwidths and kernel weights can reconstruct the omitted observations well, it is expected to provide a reasonable data-driven smoothing configuration. This idea allows the parameters to be selected without assuming that the true reliability-growth function is known.

To motivate the bandwidth-selection criterion, we begin with the integrated squared error (ISE), defined as

$$\text{ISE}(h) = \int \left\{ \hat{\Lambda}_h(t) - \Lambda(t) \right\}^2 dt, \quad (14)$$

where $\hat{\Lambda}_h(t)$ denotes the kernel estimator with bandwidth h , and $\Lambda(t)$ is the true but unknown mean value function. Minimizing the ISE yields the theoretically optimal parameter. However, since $\Lambda(t)$ is unknown, the ISE cannot be evaluated directly.

By expanding the ISE and removing terms that do not depend on the bandwidth, we obtain a computable approximation to the ISE. This leads to an ISE-inspired least-squares cross-validation (LSCV) criterion for the single-kernel estimator. In this study, the LSCV criterion is defined by

$$\text{LSCV}(h) = \int \hat{\Lambda}_h^2(t) dt - 2 \sum_{i=1}^n \left(\frac{1}{n} \sum_{\ell=1}^n \hat{\Lambda}_{(-\ell),h}(t_i) \right) \hat{\Lambda}_h(t_i), \quad (15)$$

where $\hat{\Lambda}_{(-\ell),h}(t_i)$ denotes an estimator computed after removing the ℓ -th data point. Minimizing $\text{LSCV}(h)$ provides a data-driven approximation to the ISE-optimal bandwidth.

Building on the ISE-based single-kernel framework, we extend parameter selection to the proposed multi-kernel model. In this setting, the estimator depends not only on multiple bandwidth parameters but also on the kernel weight vector $\alpha = [\alpha_1, \dots, \alpha_J]^T$, which controls the contribution of each kernel component. Therefore, parameter selection becomes a joint problem of choosing appropriate smoothing scales and determining how strongly each kernel component should contribute.

Although the number of parameters increases in the multi-kernel setting, the basic objective of parameter selection remains the same: the resulting estimator should approximate the unknown mean value function $\Lambda(t)$ as accurately as possible. Since the multi-kernel estimator is constructed through the composite kernel, the ISE-based principle used for the single-kernel estimator can be extended to the parameter pair $(\alpha, \mathbf{h})$. Accordingly, the same ISE-inspired LSCV principle is generalized to accommodate the composite kernel structure. The multi-kernel LSCV criterion is defined by

$$\text{LSCV}(\alpha, \mathbf{h}) = \int \hat{\Lambda}_{\alpha, \mathbf{h}}^2(t) dt - 2 \sum_{i=1}^n \left(\frac{1}{n} \sum_{\ell=1}^n \hat{\Lambda}_{(-\ell), \alpha, \mathbf{h}}(t_i) \right) \hat{\Lambda}_{\alpha, \mathbf{h}}(t_i), \quad (16)$$

where $\hat{\Lambda}_{(-\ell), \alpha, \mathbf{h}}(\cdot)$ is the leave-one-out estimator constructed after removing the ℓ -th data point using the composite kernel $K_{\text{MK}}(\cdot)$.

The parameter estimation problem is then formulated as the following constrained optimization problem:

$$\min_{\alpha, \mathbf{h}} \text{LSCV}(\alpha, \mathbf{h}), \quad \text{s.t. } \alpha_j \geq 0, \sum_{j=1}^J \alpha_j = 1, h_j > 0, \quad j = 1, \dots, J. \quad (17)$$

By minimizing $\text{LSCV}(\alpha, \mathbf{h})$, the kernel components with inappropriate bandwidths are automatically assigned as negligible weights, while informative smoothing scales are emphasized, leading to a data-adaptive combination of multiple scales. Fig. 1 provides an intuitive overview of the proposed kernel-based estimator configuration and LSCV-based parameter selection procedure.

The above formulation defines the LSCV-based parameter-selection criterion. The numerical implementation details, including the integration grid, bandwidth search range, optimization strategy, and repeated-run setting, are described in Section 4.4.

3.6. Prediction algorithm

This subsection describes a prediction procedure based on the proposed kernel-based estimators, including both single-kernel and multi-kernel configurations. We focus on long-horizon software reliability

prediction, where software fault counts beyond the training horizon are not observable at the time of forecasting. Accordingly, the proposed framework performs closed-loop one-step-ahead prediction based solely on information available up to the current time point. The prediction task is formulated as a sequential process over the testing period. At each step, the kernel estimator provides a fitted cumulative trend function based on the information available up to that point. The fitted cumulative value itself is not directly used as the final recursive prediction. Instead, the difference between two adjacent fitted cumulative values is interpreted as the predicted interval fault increment, and this increment is then added to the previous cumulative prediction. Therefore, the proposed procedure directly generates cumulative fault-count point predictions, which are evaluated against the observed cumulative fault counts in the testing phase.

The prediction pipeline is used for both parameter strategies considered in this paper. Specifically, prediction starts from an initial training dataset, in which model parameters are estimated using the LSCV criterion. The trained model is then applied to the testing phase, where predictions are performed in a step-by-step recursive manner.

Under the *fixed-parameter strategy*, all model parameters, including the kernel weight vector and the associated bandwidth parameters, are determined once from the training data, and remain fixed throughout the testing period. At each prediction step, the fitted cumulative trend function is computed using the fixed parameters. The recursive dataset may still be augmented by newly generated predictions, but the kernel weights and bandwidth parameters are not re-estimated. This prediction strategy is simple to implement and serves as a natural baseline for evaluating the effectiveness of more adaptive parameter updating mechanisms.

While the fixed-parameter strategy can provide stable predictions, it does not account for potential changes in the fault-detection process over time. As a result, parameters estimated from the initial training data may gradually become suboptimal as testing progresses.

To address this limitation, we further consider the *sequential updating strategy*. In this setting, parameters are re-estimated during the testing phase by incorporating newly generated cumulative predictions as pseudo-observations. Specifically, in each prediction step at r th time t_r , the time domain is re-normalized to the unit interval $[0, 1]$ relative to the current observation window. The kernel weights and bandwidth parameters are then updated using the recursively constructed dataset available up to the current step.

It is important to note that, for long-term prediction, the model uses its own predicted values to forecast subsequent steps. While this recursive structure may lead to error accumulation, the sequential updating strategy provides a recalibration process based on the recursively constructed prediction trajectory. By recalibrating bandwidths and weights via LSCV using the latest recursively constructed dataset, the framework may help reduce the drift effects typically associated with static recursive prediction.

Both parameter estimation strategies follow the same prediction pipeline and can be applied to both single-kernel and multi-kernel estimators. The only difference is whether the model parameters are updated during the testing phase. This unified formulation enables a fair comparison between the fixed-parameter strategy and the sequential updating strategy in the evaluation.

Increment-based recursive update and monotonicity. It should be noted that the fitted cumulative trend produced by the kernel estimator and the final recursive prediction are not identical. At each prediction step, the estimator first provides fitted cumulative trend values at two adjacent time points, denoted by

$$\tilde{m}_r = \hat{\Lambda}_r(t_r), \quad \tilde{m}_{r+1} = \hat{\Lambda}_r(t_{r+1}).$$

Their difference is interpreted as the predicted interval fault increment:

$$\hat{d}_{r+1} = \tilde{m}_{r+1} - \tilde{m}_r.$$

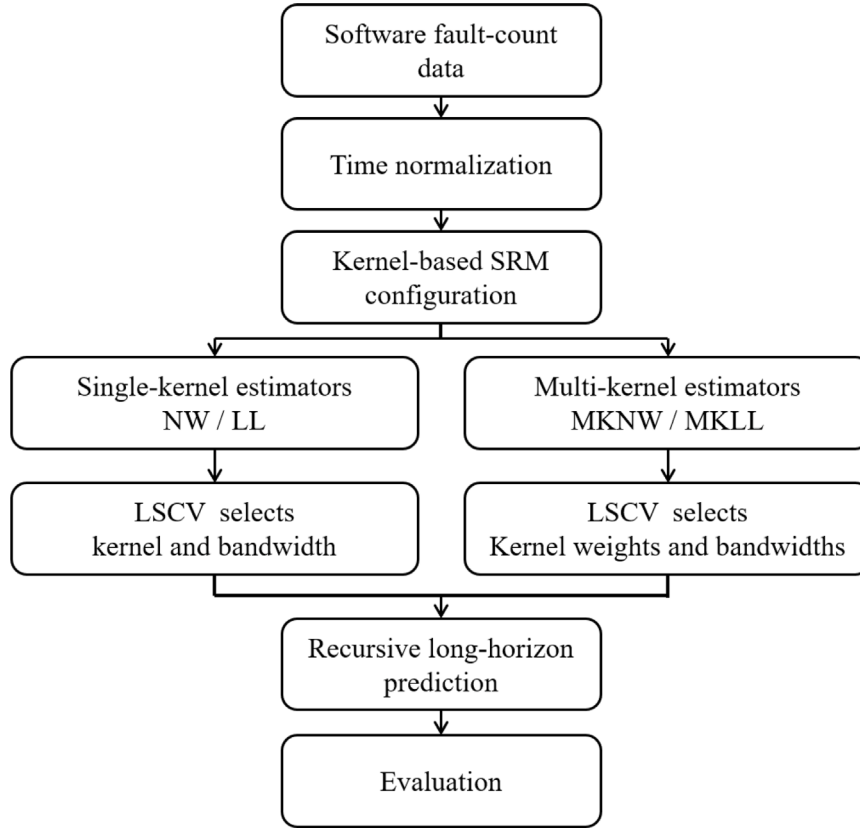


Fig. 1. Overview of single-kernel and multi-kernel estimator configuration with LSCV-based parameter selection.

Since interval fault increments should be nonnegative, we apply the correction

$$\hat{d}_{r+1}^+ = \max(0, \hat{d}_{r+1}).$$

The final cumulative prediction is then updated recursively as

$$\hat{y}_{r+1} = \hat{y}_r + \hat{d}_{r+1}^+,$$

where $\hat{y}_r$ denotes the cumulative fault-count prediction available from the previous step. Therefore, the final predicted cumulative sequence is nondecreasing over time, even though the unconstrained kernel fitted curve itself is not theoretically guaranteed to be monotone. This correction does not use future observed fault counts; it only enforces the basic nonnegativity property of interval fault increments.

To make the recursive prediction procedure more intuitive, we consider a toy cumulative fault-count sequence:

$$\{(t_i, N(t_i))\}_{i=1}^8 = \{(1, 2), (2, 5), (3, 9), (4, 12), (5, 14), (6, 16), (7, 17), (8, 18)\}.$$

Each pair $(t_i, N(t_i))$ represents the cumulative number of detected faults $N(t_i)$ at time index t_i . In this example, the first four observations are used as the initial training data, while the remaining four observations represent the prediction horizon. During prediction, the future observed cumulative fault counts, such as $N(5) = 14$ and $N(6) = 16$, are not used by the model; they are shown only to clarify the data structure (see Table 3).

While Section 3 presents the methodological formulation of the proposed framework, Section 4 translates these algorithms into an empirical framework for real-world validation.

4. Experimental setup

4.1. Research questions

We evaluate the proposed nonparametric prediction framework through an empirical study on multiple software fault-count datasets.

Algorithm 1: Reliability Prediction under the Fixed-Parameter Strategy and Sequential Updating Strategy

Input: Training data D_{train} , forecasting horizon T , and strategy $S \in \{F, D\}$

Output: Predicted cumulative fault-count sequence $\{\hat{y}_r\}_{r=1}^T$

Initialize $D_0 \leftarrow D_{\text{train}}$;

Set t_0 and $\hat{y}_0$ as the last time point and the last cumulative fault count in D_{train} ;

Estimate the initial parameters θ_0 from D_0 using the LSCV criterion;

for $r = 0$ **to** $T - 1$ **do**

 Re-normalize the time indices in the current working dataset D_r and the next prediction time point to the unit interval $[0, 1]$;

 Construct the fitted cumulative trend function $\hat{\Lambda}_r(\cdot)$ using D_r and θ_r ;

 Compute the fitted trend values $\tilde{m}_r = \hat{\Lambda}_r(t_r)$ and $\tilde{m}_{r+1} = \hat{\Lambda}_r(t_{r+1})$;

 Compute the predicted increment $\hat{d}_{r+1} = \tilde{m}_{r+1} - \tilde{m}_r$;

 Apply the nonnegative correction $\hat{d}_{r+1}^+ = \max(0, \hat{d}_{r+1})$;

 Update the cumulative prediction $\hat{y}_{r+1} = \hat{y}_r + \hat{d}_{r+1}^+$;

 Augment the working dataset with the newly predicted point

$D_{r+1} = D_r \cup \{(t_{r+1}, \hat{y}_{r+1})\}$;

if $S = D$ **then**

 Re-estimate θ_{r+1} from D_{r+1} using the LSCV criterion;

else

 Keep the parameters fixed, i.e., $\theta_{r+1} \leftarrow \theta_r$;

The experiments are designed to assess point-prediction accuracy and to clarify the roles of the sequential updating strategy and different model configurations in long-horizon software reliability prediction. Accordingly, we address the following research questions.

• RQ1: Comparative Point-Prediction Accuracy.

How does the proposed nonparametric framework perform compared with representative NHPP-based, wavelet-based, and LSTM-based baselines in terms of point-prediction accuracy?

Table 3
A toy example of recursive prediction.

Stage	Fixed-parameter strategy (F)	Sequential updating strategy (D)
Initial training	Use $D_4 = \{(t_i, N(t_i))\}_{i=1}^4 = \{(1, 2), (2, 5), (3, 9), (4, 12)\}$. Select parameters by LSCV and keep them fixed.	Use the same D_4 . Select initial parameters by LSCV.
Predict time 5	Compute $\tilde{m}_4, \tilde{m}_5$, obtain $\hat{d}_5^+ = \max(0, \tilde{m}_5 - \tilde{m}_4)$, and update $\hat{y}_5 = 12 + \hat{d}_5^+$.	Same computation: $\hat{d}_5^+ = \max(0, \tilde{m}_5 - \tilde{m}_4)$, $\hat{y}_5 = 12 + \hat{d}_5^+$.
Before time 6	Append the newly predicted point $(5, \hat{y}_5)$: $D_5 = D_4 \cup \{(5, \hat{y}_5)\}$. Parameters remain fixed.	Append the newly predicted point $(5, \hat{y}_5)$: $D_5 = D_4 \cup \{(5, \hat{y}_5)\}$. Re-estimate parameters by LSCV.
Predict time 6	Using fixed parameters, obtain $\hat{d}_6^+$ from the fitted trend difference and update $\hat{y}_6 = \hat{y}_5 + \hat{d}_6^+$.	Using updated parameters, obtain $\hat{d}_6^+$ from the fitted trend difference and update $\hat{y}_6 = \hat{y}_5 + \hat{d}_6^+$.
Continue	Append each newly predicted point, but keep parameters fixed.	Append each newly predicted point and update parameters recursively.

This question examines whether the proposed nonparametric approach can reduce point-prediction error relative to representative baselines, especially in long-horizon forecasting settings where the actual fault-detection trend may deviate from the assumed parametric mean value function.

• **RQ2: Effect of the Sequential Updating Strategy.**

Does the sequential updating strategy improve long-horizon point prediction compared with the fixed-parameter strategy?

To separate the effect of the kernel estimator from the effect of the parameter-updating strategy, we compare fixed-parameter kernel estimators, such as SKLL-F and MKLL-F, with their sequential counterparts, such as SKLL-D and MKLL-D. The sequential variants use model-generated cumulative predictions as pseudo-observations only to construct a long-horizon forecasting trajectory. These generated values are not interpreted as actual observed fault counts.

• **RQ3: Model Configuration and Empirical Stability.**

Which estimator and kernel configuration yield stable and reliable predictions in practice?

This question focuses on practical modeling choices, including the comparison between NW and LL estimators and between single-kernel and multi-kernel constructions. Since the optimization of multi-kernel weights and bandwidths involves random initialization, we further repeat all multi-kernel experiments using 20 independent random seeds. The averaged PMAE values are reported as the main results, and the standard deviations across repeated runs are used to assess the sensitivity of the multi-kernel optimization procedure to random initialization.

4.2. Datasets

To assess the point-prediction performance of the proposed framework under different software development contexts, we conduct empirical experiments on sixteen real-world fault-count datasets. These datasets are drawn from two sources. The first group consists of eight closed-source software systems, denoted as CSS 1–CSS 8, which have been widely used in prior software reliability studies and enable comparison with existing models. The second group includes eight open-source software datasets, denoted as OSS 1–OSS 8, constructed from GitHub repository records. For the OSS datasets, a fault-related record is defined as a commit or issue record that satisfies the fault-fix identification criteria described in the replication package. These criteria are based on fault-related keywords and repository metadata associated with bug fixing, error correction, defect repair, crash resolution, or similar corrective maintenance activities. The selected records are sorted chronologically and aggregated into fixed monthly intervals to construct grouped fault-count data. The interval fault counts are then accumulated over time to obtain the cumulative fault-count sequences used in the experiments. These OSS datasets provide complementary evaluation settings that reflect modern development practices and evolving system complexity. Several closed-source datasets contain

only a limited number of observation intervals. Therefore, the results on these datasets are mainly intended to evaluate point-prediction behavior under small-sample conditions. To examine empirical stability, we report the results separately for each dataset and testing phase, rather than relying solely on aggregated averages. The CSS and OSS datasets are summarized in Tables 4 and 5, respectively. To support reproducibility, the replication package includes the processed datasets, OSS raw records, data-construction scripts, experimental scripts and instructions for reproducing the reported PMAE tables.

4.3. Benchmark methods and model selection

To evaluate the proposed nonparametric approach, we consider three types of baseline methods: traditional NHPP-based SRMs, a recent wavelet-based prediction model [19], and LSTM-based neural prediction models [16]. For the NHPP-based baseline, all representative mean value functions (MVFs) listed in Table 1 are fitted to the initial training data by maximum likelihood estimation. The SRM with the lowest AIC is selected as the fixed parametric baseline.

The wavelet-based baseline is implemented using the tool released by the original study, following its recommended settings. For the LSTM-based baseline, we adopt a recursive one-step-ahead prediction strategy. The model is trained using only the initial training data. During the testing phase, after each prediction step, the predicted cumulative fault count is appended to the input sequence and used to predict the next step. No future observed fault counts from the testing horizon are used during this recursive forecasting process. This setting makes the LSTM baseline consistent with the long-horizon forecasting protocol used for the proposed kernel-based models, while avoiding unstable direct multi-step prediction from short cumulative fault-count sequences.

For the single-kernel nonparametric models, we evaluate a candidate set of four kernel functions, and the kernel type and bandwidth are selected according to the LSCV-based procedure described in Section 4.4. For the proposed multi-kernel models, the kernel structure is determined through weight optimization rather than by selecting a single kernel function. The mixing weights of the candidate kernels and their associated bandwidths are optimized to allow the composite kernel to adapt to heterogeneous fault-detection patterns. Because the constrained optimization problem in the multi-kernel models involves randomized initialization, all multi-kernel experiments are repeated 20 times using independent random seeds. For each dataset, testing phase, and multi-kernel model, PMAE is computed for each run, and the mean PMAE over the 20 runs is used as the primary reported value. The standard deviation across the 20 runs is also recorded to evaluate the sensitivity of the multi-kernel optimization procedure to random initialization. The full mean-and-standard-deviation results are reported in Appendix Tables B.14–B.16.

Table 6 summarizes the models included in the comparative study, including their model classes, selection strategies, updating policies, and experimental roles.

Table 4
Summary of closed-source software fault-count datasets.

ID	Length	No. Faults	Source	Nature of System
CSS 1	17	54	SYS2 [34]	Real-time command and control system
CSS 2	14	38	SYS3 [34]	Real-time command and control system
CSS 3	19	120	Release2 [35]	Tandem software system
CSS 4	14	9	NASA [36]	Inertial navigating system
CSS 5	20	66	CSS 1 [37]	Embedded application for printer
CSS 6	30	52	CSS 3 [37]	Embedded application for printer
CSS 7	181	224	J2 [1]	JPL Galileo planetary orbiter software system
CSS 8	114	188	J4 [1]	Anonymized JPL spacecraft software system with AACS and CDS subsystems

Table 5
Summary of open-source software fault-count datasets collected from GitHub repositories.

ID	Length	No. Faults	Source	Nature of System
OSS 1	133	150	Redis [38]	High-performance in-memory key-value data storage system
OSS 2	111	371	Wox [39]	Standalone desktop utility application
OSS 3	148	190	Backbone [40]	Client-side web application framework for frontend development
OSS 4	85	768	Homebrew [41]	System-level package management software interacting with OS
OSS 5	79	1266	PyTorch [42]	Large-scale deep learning and numerical computation framework
OSS 6	121	2069	Chart.js [43]	JavaScript library for web-based data visualization
OSS 7	113	234	DevDocs [44]	Web-based API documentation browser for software developers
OSS 8	61	1089	RSSHub [45]	RSS feed generation and content aggregation service

Table 6
Summary of benchmark methods and investigated models.

Model	Model Class	Prediction/Updating Policy	Configuration
NHPP	Parametric SRM	Fixed-parameter baseline	Ten candidate MVFs are fitted to the initial training data by MLE, and the MVF with the lowest AIC is selected
Wavelet	Signal-processing model	Original setting	Implemented using the tool released by the original study with its recommended settings
LSTM	Neural network model	Recursive one-step-ahead prediction	Recursively predicts future cumulative fault counts by feeding model-generated predictions back into the input sequence
SKNW/SKLL	Single-kernel models	Fixed-parameter strategy (F)/Sequential updating strategy (D)	NW/LL estimators with LSCV-selected kernel and bandwidth; D updates bandwidth recursively
MKNW/MKLL	Multi-kernel models	Fixed-parameter strategy (F)/Sequential updating strategy (D)	Multi-kernel NW/LL with optimized weights and bandwidths; D updates parameters recursively

Note: F = fixed-parameter strategy; D = sequential updating strategy.

4.4. Implementation details of kernel models

All kernel-based models are implemented on the normalized time domain. For each dataset and training cutoff, the available time points are transformed to the unit interval. The same normalization rule is applied to the time indices of future prediction points, but no future fault-count values are used during prediction. This normalization makes bandwidth values comparable across datasets with different observation lengths.

For the single-kernel models, four candidate kernels are considered: Gaussian, triangular, biweight, and Epanechnikov kernels. For each candidate kernel, the bandwidth parameter is selected by minimizing the LSCV criterion on the current working dataset. The LSCV integral term is computed by the trapezoidal rule on a uniform grid of 100 points over the normalized time domain. The bandwidth search is conducted over a bounded interval $h \in [h_{\min}, 1.0]$, where $h_{\min} = \min\{0.95, \max(0.05, 3/n)\}$ on the normalized time domain and n denotes the number of observations in the current working dataset. This setting is used as a numerical safeguard rather than as a dataset-specific tuning rule. The term $3/n$ prevents the optimizer from selecting extremely small bandwidths relative to the average spacing of observations, while the constant lower bound 0.05 avoids degenerate local fits on the normalized time scale. The cap of 0.95 on $h_{\min}$ ensures that the search interval remains valid even under very small training samples. The same rule is applied consistently to all datasets, testing phases, and kernel-based models. The candidate kernel and bandwidth with the smallest LSCV value are selected for prediction.

For the multi-kernel models, the four kernels are used simultaneously as base kernels. Each base kernel has its own bandwidth, and the final composite kernel is represented as a convex combination of the base kernels. The kernel weights are constrained to be nonnegative and to sum to one. This constraint is implemented through a softmax parameterization of the weight vector. Thus, the optimization variables include the unconstrained weight parameters and the bandwidths of the base kernels. The same numerical integration rule and bounded bandwidth-search strategy are used for the multi-kernel LSCV criterion.

The multi-kernel optimization problem is solved using particle swarm optimization with a swarm size of 50 and a maximum of 50 iterations. Invalid objective values, such as NaN or infinite values, are assigned a large penalty value during optimization. To reduce the influence of local optima and randomized initialization, each multi-kernel experiment is repeated using 20 independent random seeds. For each dataset, testing phase, and multi-kernel model, PMAE is computed for each run, and the mean PMAE over the 20 runs is reported in the main result tables. The corresponding standard deviations are provided in Appendix Tables B.14–B.16. The detailed implementation, including the bandwidth-bound setting, is provided in the replication package.

4.5. Training and testing strategy

To examine predictive performance at different stages of the software testing process, we adopt a time-ordered training–testing scheme. Specifically, we use three training cutoffs at 20%, 50%, and 80% of

the observation intervals in each dataset, corresponding to the early, middle, and late testing phases, respectively.

Let each dataset consist of n observation intervals. For each cutoff, a model is fitted using data from the first k intervals, where

$$k \in \{[0.2n], [0.5n], [0.8n]\}.$$

The fitted model is then used to predict the cumulative number of detected software faults over the remaining $n-k$ intervals. Performance is evaluated exclusively on these future intervals to assess predictive ability on unseen data. For the sequential updating kernel variants, model-generated cumulative predictions are used only to construct a model-based forecasting trajectory and are not interpreted as actual observed fault counts.

4.6. Evaluation metrics

Prediction Mean Absolute Error (PMAE) is used as the primary metric for evaluating long-horizon prediction performance. Because the datasets differ considerably in scale, we also report two count-based summary statistics to provide a complementary empirical comparison across datasets.

4.6.1. Primary accuracy metric: PMAE

PMAE quantifies the average deviation between the predicted and observed cumulative fault counts over the *testing horizon*, and is defined by

$$\text{PMAE} = \frac{1}{n-k} \sum_{i=k+1}^n |N(t_i) - \hat{y}_i|, \quad (18)$$

where;

- n is the total number of observation intervals (dataset length),
- k is the index of the last interval used for training,
- $N(t_i)$ represents the observed cumulative fault count at time t_i ,
- $\hat{y}_i$ is the final predicted cumulative fault count at time t_i obtained by the recursive prediction procedure using the training set $\{(t_j, N(t_j))\}_{j=1}^k$.

A lower PMAE indicates better long-horizon point-prediction accuracy for cumulative fault-count prediction.

4.6.2. Summary statistics for model robustness

Due to the substantial variation in dataset scale, direct comparison of raw PMAE values across datasets can be misleading. To provide a scale-independent and interpretable assessment, we report two count-based indicators.

(1) *best-performance count* (N_{best}). This statistic counts how often a model achieves the lowest PMAE among all models in the comparison;

$$N_{\text{best}}(m) = \sum_{d \in S} \mathbb{I} \left(\text{PMAE}_{m,d} = \min_{m' \in \mathcal{A}} \text{PMAE}_{m',d} \right). \quad (19)$$

where S denotes the set of datasets and $\mathcal{A}$ denotes the set of compared models. A higher N_{best} indicates that a model more frequently attains the best PMAE across datasets.

(2) *outperformance count against the baseline* (N_{better}). This measure evaluates how consistently a model outperforms the AIC-selected parametric NHPP baseline on each dataset;

$$N_{\text{better}}(m) = \sum_{d \in S} \mathbb{I} \left(\text{PMAE}_{m,d} < \text{PMAE}_{\text{Base},d} \right). \quad (20)$$

Here, $\text{PMAE}_{\text{Base},d}$ denotes the PMAE of the AIC-selected NHPP baseline on dataset d . A higher N_{better} indicates that a model more frequently achieves lower PMAE than the AIC-selected fixed NHPP baseline. Collectively, PMAE, N_{best} , and N_{better} provide an empirical summary of point-prediction accuracy and dataset-wise consistency across heterogeneous datasets.

5. Experimental results

This section presents an empirical evaluation of the proposed non-parametric framework for long-horizon software reliability prediction. Based on PMAE values obtained from sixteen datasets across three testing phases, we examine how model structure and the sequential updating strategy influence point-prediction performance.

5.1. Overall performance

Tables 7 summarize the PMAE values of the AIC-selected fixed NHPP baseline, wavelet-based predictor, LSTM-based predictor, and eight kernel-based nonparametric estimators across the three testing phases. For the multi-kernel models, the reported values are mean PMAE values obtained from 20 independent runs with different random seeds. This repeated-run setting was introduced to reduce the influence of random initialization in the constrained optimization of kernel weights and bandwidths. The corresponding standard deviations are generally small in most cases, with a few larger values observed for specific dataset-model combinations, as reported in Appendix Tables B.14–B.16. To provide a dataset-wise summary of relative performance, Table 10 reports two count-based indicators, N_{best} and N_{better} .

5.1.1. Early testing phase

Table 7 presents the prediction results when the models are trained using only 20% of the available observations. In this phase, the models must extrapolate over a long prediction horizon from limited training data. Consequently, prediction errors can accumulate during recursive forecasting, making model configuration particularly important.

The proposed kernel-based methods demonstrate strong predictive performance in the early testing phase. In particular, MKLL-D achieves the lowest PMAE on CSS 7, CSS 8, OSS 2, OSS 6, and OSS 7, giving it the highest N_{best} value of 5 in this phase. Moreover, MKLL-D outperforms the classical NHPP baseline on 14 of the 16 datasets. These results indicate that the combination of the local-linear estimator and sequential updating strategy is particularly effective when only limited early-stage observations are available.

For example, on OSS 2, NHPP yields a PMAE of 100.5788, whereas MKLL-D reduces the PMAE to 46.1683. On OSS 6, MKLL-D reduces the PMAE from 948.7637 for NHPP to 675.1136, slightly outperforming LSTM, which obtains a PMAE of 677.9059. Nevertheless, the results also show that increased model complexity does not guarantee better performance under all data conditions. For instance, SKLL-D achieves the lowest PMAE on OSS 8, while Wavelet and LSTM remain competitive on several other datasets.

Fig. 2 illustrates the prediction trajectories on OSS 2 under the early observation setting. Because only the initial 20% of the fault-count process is observed, the remaining portion constitutes a challenging long-horizon prediction task. NHPP, Wavelet, and LSTM produce relatively flat trajectories and underestimate the subsequent growth in cumulative fault counts. By contrast, the LL-based kernel models, particularly SKLL-D and MKLL-D, generate steeper trajectories and follow the observed growth more closely. This visual pattern is consistent with the quantitative results in Table 7, where MKLL-D achieves the lowest PMAE on OSS 2. Nevertheless, no single model dominates all datasets, confirming that predictive effectiveness remains dependent on the underlying fault-detection characteristics.

5.1.2. Middle testing phase

When the training cutoff increases to 50% (8), the prediction task shifts from severe data sparsity toward capturing more complex and potentially non-stationary fault-growth patterns. The additional observations generally improve the characterization of the underlying growth trend, although the relative effectiveness of different model configurations remains dataset-dependent.

Table 7
Comparison of PMAE values in early testing phase.

Dataset	Baseline			Single-kernel				Multi-kernel			
	NHPP	Wavelet	LSTM	SKNW-F	SKNW-D	SKLL-F	SKLL-D	MKNW-F	MKNW-D	MKLL-F	MKLL-D
<i>Closed-source datasets</i>											
CSS 1	26.6183 (exp)	23.8143	27.0653	32.4476 (bi)	32.9488 (bi)	28.0775 (Ep)	27.6278 (Ep)	32.4475	32.9487	28.0770	27.1626
CSS 2	13.8491 (gamma)	12.6233	8.1917	7.7161 (bi)	9.1675 (bi)	10.8773 (Ga)	10.8773 (Ga)	9.8581	10.0569	7.2382	9.6456
CSS 3	31.2307 (exp)	10.2303	10.3011	45.6342 (bi)	48.6770 (bi)	15.1024 (Ep)	14.5493 (Ep)	45.6332	48.6405	15.1011	11.8234
CSS 4	0.3385 (exp)	0.3073	1.9086	2.9790 (bi)	3.1668 (bi)	0.2640 (Ep)	0.2852 (Ep)	3.3622	3.2911	0.2662	0.3068
CSS 5	17.9001 (lnorm)	16.0764	17.6480	10.1271 (bi)	12.4459 (bi)	34.4418 (Ga)	34.4418 (Ga)	10.1262	12.4455	34.4389	33.4714
CSS 6	31.8512 (tlogist)	10.4480	23.1483	32.0182 (tr)	32.2808 (tr)	23.0662 (tr)	21.3779 (tr)	32.0183	32.4857	23.0663	20.9457
CSS 7	64.3549 (lnorm)	71.5633	91.8044	130.7910 (bi)	130.8557 (bi)	45.0911 (Ep)	41.8129 (Ep)	130.7993	130.8611	45.4419	41.7495
CSS 8	30.0852 (tnorm)	32.5932	61.6269	73.1821 (tri)	73.3102 (tri)	25.2744 (Ep)	24.6372 (Ep)	73.1846	73.8230	25.2706	24.6129
<i>Open-source datasets</i>											
OSS 1	41.1298 (exp)	63.8308	7.7248	43.4340 (bi)	43.4340 (bi)	528.7653 (bi)	43.4340 (bi)	43.4340	43.4339	20.1036	14.0345
OSS 2	100.5788 (exp)	129.7788	137.7728	177.8053 (tri)	178.0387 (tri)	55.1309 (Ep)	46.2747 (Ep)	177.8064	179.4976	55.4351	46.1683
OSS 3	693.6397 (tlogist)	158.3926	408.3635	528.7653 (Ep)	25.1283 (bi)	454.3391 (Ep)	528.7653 (Ep)	24.6468	25.1398	452.4268	530.4164
OSS 4	177.7236 (llogist)	34.7203	139.6634	179.6672 (bi)	179.8958 (bi)	138.7566 (bi)	143.2968 (bi)	179.6673	179.8958	138.7527	143.2384
OSS 5	426.4058 (llogist)	356.5592	337.6318	436.1083 (bi)	436.3909 (bi)	346.8557 (Ga)	355.3475 (Ga)	436.1082	436.3911	358.7471	369.9175
OSS 6	948.7637 (tlogist)	816.9800	677.9059	980.3793 (bi)	980.8362 (bi)	700.8685 (Ep)	675.5764 (Ep)	980.3790	980.8453	699.9824	675.1136
OSS 7	108.8664 (lxvmax)	82.2711	102.5197	113.5602 (tri)	113.5964 (tri)	74.6454 (bi)	65.7820 (bi)	113.5603	113.8205	78.2788	64.3596
OSS 8	267.8013 (lxvmax)	272.5658	222.5622	344.4780 (bi)	345.9005 (bi)	212.8294 (Ep)	175.5490 (Ep)	344.4777	345.9005	212.8358	175.8876

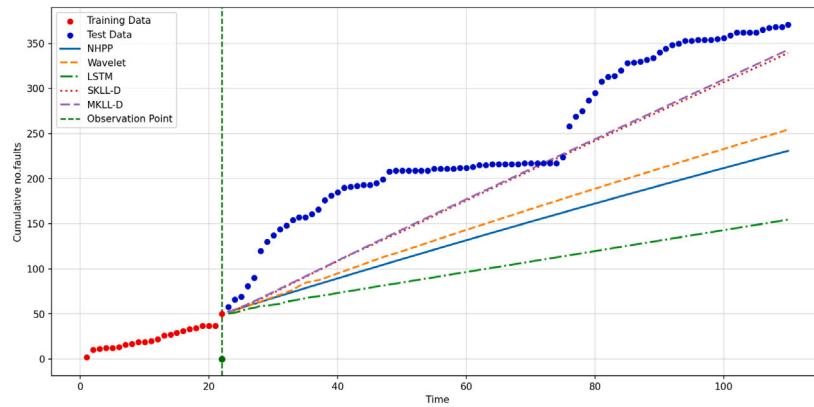


Fig. 2. Behavior of predictive performances in early testing phase (20%).

The MKLL-F model produces a lower PMAE than MKLL-D on 10 of the 16 datasets, whereas MKLL-D performs better on the remaining six datasets. MKLL-F achieves the best overall result on CSS 5 and OSS 6, while MKLL-D ranks first on CSS 2 and OSS 2. Among all models, SKLL-F and LSTM each obtain the lowest PMAE on three datasets. These results indicate that the benefit of sequential updating strategy becomes less uniform after a larger portion of the fault-growth process has been observed.

Kernel-based models nevertheless remain competitive with the classical NHPP baseline. On OSS 2, for example, MKLL-D reduces the PMAE from 77.9153 for NHPP to 19.9595. On OSS 6, MKLL-F achieves the lowest PMAE of 113.8614, compared with 474.6778 for NHPP. Strong

results are also obtained by the single-kernel LL estimator, with SKLL-F attaining the lowest PMAE on OSS 1, OSS 4, and OSS 7. Conversely, NHPP performs best on CSS 8 and OSS 3, while LSTM ranks first on CSS 7, OSS 5, and OSS 8. Thus, no individual model dominates all datasets in the middle phase.

Fig. 3 illustrates the prediction trajectories on OSS 2 under the middle observation setting. With 50% of the observations available for training, the models have more information about the fault-growth trend than in the early phase, but must still predict a substantial increase after the observation point. In this representative case, MKLL-D provides the lowest PMAE among the evaluated models, which is consistent with its closer tracking of the subsequent cumulative fault

Table 8
Comparison of PMAE values in middle testing phase.

Dataset	Baseline			Single-kernel				Multi-kernel			
	NHPP	Wavelet	LSTM	SKNW-F	SKNW-D	SKLL-F	SKLL-D	MKNW-F	MKNW-D	MKLL-F	MKLL-D
<i>Closed-source datasets</i>											
CSS 1	4.3516 (tlogist)	8.9899	17.1209	1.6617 (bi)	1.9735 (bi)	15.9246 (Ep)	17.1042 (Ep)	1.6616	1.9733	15.9809	17.1097
CSS 2	7.1373 (lxvmax)	2.2531	8.4051	7.3505 (bi)	7.4337 (bi)	1.3116 (Ga)	1.3116 (Ga)	7.3504	7.4336	1.3109	1.0424
CSS 3	20.3933 (exp)	7.7576	31.4964	10.6251 (bi)	11.0851 (bi)	19.7678 (Ep)	22.5296 (Ep)	10.6246	11.0847	19.8558	22.5577
CSS 4	0.4153 (exp)	0.3861	0.4395	1.3371 (bi)	1.3922 (bi)	0.7377 (Ep)	0.9143 (Ep)	1.3371	1.3922	0.7237	0.9217
CSS 5	18.5511 (lxvmax)	14.8857	12.6606	17.6952 (bi)	17.7871 (bi)	12.4916 (bi)	12.6402 (bi)	17.6952	17.7871	10.3381	10.4119
CSS 6	11.4663 (tlogist)	12.1755	155.9284	6.2219 (bi)	5.7527 (bi)	54.4308 (Ep)	63.2720 (Ep)	6.2226	5.7533	54.5453	63.2618
CSS 7	249.8076 (tlogist)	19.1502	18.1487	54.8721 (bi)	54.8721 (bi)	83.2366 (Ep)	82.5965 (Ep)	54.9984	54.9102	83.2048	82.5734
CSS 8	11.6483 (lxvmin)	50.4835	13.8837	56.8941 (tri)	56.9121 (tri)	41.9134 (Ep)	41.1717 (Ep)	57.2752	58.4753	37.4788	40.0515
<i>Open-source datasets</i>											
OSS 1	9.4669 (lxvmax)	9.3704	28.4676	9.9706 (bi)	9.9706 (bi)	4.1622 (bi)	33.7706 (bi)	9.9899	9.9763	30.1002	39.9439
OSS 2	77.9153 (tlogist)	65.4647	26.6129	79.7744 (bi)	79.7938 (bi)	23.9646 (Ep)	21.5573 (Ep)	79.7422	79.7926	23.4027	19.9595
OSS 3	3.2811 (tnorm)	18.0348	70.6086	4.1622 (bi)	4.1622 (bi)	4.1622 (bi)	4.1622 (bi)	4.1689	4.1663	10.7960	5.3858
OSS 4	226.1623 (lxvmax)	223.6473	215.8272	240.4922 (bi)	240.5545 (bi)	210.5987 (bi)	211.1062 (bi)	240.4964	240.5553	210.6117	210.8133
OSS 5	422.6838 (tlogist)	271.9902	51.6807	383.1706 (bi)	383.6163 (bi)	77.6060 (Ep)	67.7341 (Ep)	383.6354	383.6635	76.7107	68.1119
OSS 6	474.6778 (tlogist)	317.1189	253.4189	503.7588 (bi)	503.7588 (bi)	113.8714 (Ep)	173.5923 (Ep)	503.9031	503.9207	113.8614	171.4229
OSS 7	37.0408 (tlogist)	20.5397	18.4421	54.4696 (bi)	54.5171 (bi)	9.5959 (Ep)	12.0200 (Ep)	54.3004	54.3837	9.7628	12.0411
OSS 8	236.2430 (tlogist)	200.0367	121.5765	283.6044 (bi)	283.9876 (bi)	143.9757 (Ep)	140.2214 (Ep)	283.6183	283.9925	143.5829	140.0602

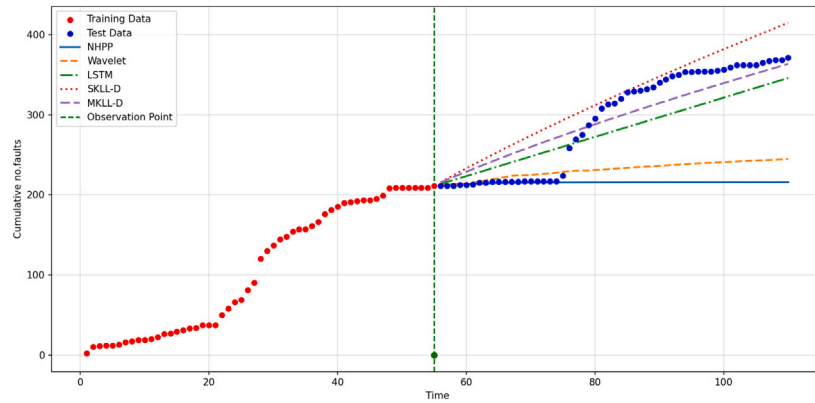


Fig. 3. Prediction trajectories on OSS 2 in the middle testing phase using 50% of the observations for training.

growth. However, the varying results across the 16 datasets confirm that the benefit of sequential bandwidth updating depends on the characteristics of the underlying fault-detection process.

5.1.3. Late testing phase

In the late testing phase, as shown in 9, the models have access to 80% of the fault-count observations. The remaining prediction horizon is therefore shorter than those in the early and middle phases. Nevertheless, the relative performance of the models remains dependent on the fault-growth characteristics of each dataset.

NHPP achieves the lowest PMAE on five datasets, namely CSS 4, CSS 5, CSS 7, OSS 3, and OSS 7, giving it the highest N_{best} value in the late phase. This result indicates that conventional parametric models can remain highly competitive when the observed fault-growth process is consistent with an appropriate mean value function.

LSTM obtains the lowest PMAE on four datasets: CSS 6, OSS 2, OSS 4, and OSS 5. Its strong performance suggests that the additional observations available in the late phase enable the network to learn the temporal fault-growth pattern more effectively. For example, on OSS 2, LSTM achieves a PMAE of 3.1643, compared with 19.4657 for NHPP and 10.0560 for Wavelet.

The LL-based kernel models also remain competitive. MKLL-D achieves the lowest PMAE on CSS 8 and OSS 1, SKLL-D performs best on CSS 2, SKLL-F ranks first on CSS 3, and MKLL-F obtains the best result on OSS 8. In addition, Wavelet achieves the lowest PMAE on CSS 1 and OSS 6. These results demonstrate that no single modeling paradigm dominates the late phase, although NHPP and LSTM obtain the largest numbers of best-performing cases.

Overall, the late-phase results differ from those observed in the early phase. Sequential multi-kernel updating is less dominant when most

Table 9
Comparison of PMAE values in late testing phase.

Dataset	Baseline			Single-kernel				Multi-kernel			
	NHPP	Wavelet	LSTM	SKNW-F	SKNW-D	SKLL-F	SKLL-D	MKNW-F	MKNW-D	MKLL-F	MKLL-D
<i>Closed-source datasets</i>											
CSS 1	4.0338 (tlogist)	0.4765	9.8737	3.5225 (bi)	3.5296 (bi)	3.4093 (bi)	3.4093 (bi)	3.5224	3.5295	0.4858	0.6623
CSS 2	1.3894 (lxvmax)	1.2250	1.6367	1.5407 (bi)	1.5537 (bi)	0.8042 (bi)	0.7852 (bi)	1.5408	1.5537	0.9584	0.9573
CSS 3	1.2995 (tnorm)	12.5875	2.5591	2.0916 (bi)	2.0904 (bi)	0.3382 (bi)	0.3397 (bi)	2.0915	2.0903	3.6493	3.4235
CSS 4	0.3143 (exp)	0.8078	0.5018	0.5889 (bi)	0.5812 (bi)	0.3181 (Ga)	0.3181 (Ga)	0.5889	0.5812	0.3181	0.3471
CSS 5	1.9676 (lxvmax)	4.5916	3.9257	1.9940 (bi)	2.0053 (bi)	8.6654 (bi)	7.9480 (bi)	1.9939	2.0052	8.3023	7.7477
CSS 6	2.2613 (tlogist)	4.9042	0.5619	2.3333 (bi)	2.3333 (bi)	5.9998 (Ep)	4.2068 (Ep)	2.3333	2.3333	2.2391	1.9542
CSS 7	1.9053 (tnorm)	11.6205	11.4791	2.7267 (bi)	2.7267 (bi)	24.7537 (Ep)	29.2698 (Ep)	2.7358	2.7310	24.6093	28.3071
CSS 8	10.8855 (tlogist)	6.3507	6.6498	19.0925 (bi)	19.0925 (bi)	4.0544 (Ep)	2.9715 (Ep)	19.0955	19.1036	4.0654	2.9596
<i>Open-source datasets</i>											
OSS 1	2.2001 (lxvmax)	8.1757	8.0419	7.4231 (bi)	7.4231 (bi)	7.4231 (bi)	7.4231 (bi)	7.4238	7.4234	2.1687	2.1204
OSS 2	19.4657 (gamma)	10.0560	3.1643	20.4708 (bi)	20.4708 (bi)	37.2427 (Ep)	53.8206 (Ep)	20.4776	20.4920	37.7164	53.6525
OSS 3	0.8405 (tlogist)	1.0046	2.1822	1.0786 (bi)	0.9697 (bi)	1.0786 (bi)	1.0786 (bi)	0.9699	0.9698	1.0766	1.0750
OSS 4	44.2598 (tlogist)	89.7251	18.6038	151.0374 (bi)	151.0374 (bi)	38.1605 (Ep)	49.7883 (Ep)	151.0867	151.1005	39.3875	48.7939
OSS 5	162.0676 (tlogist)	144.6789	87.2121	212.8260 (bi)	212.8260 (bi)	98.2932 (Ep)	96.7689 (Ep)	212.9208	212.8908	98.1103	96.6448
OSS 6	79.9252 (lnorm)	60.9137	83.0272	231.7387 (bi)	231.7387 (bi)	83.8271 (Ep)	83.8271 (Ep)	231.8644	231.8751	82.6313	83.4414
OSS 7	2.3189 (txvmax)	5.4145	12.4341	11.1033 (bi)	11.1033 (bi)	5.3684 (Ep)	5.3684 (Ep)	11.1114	11.1177	5.4514	5.3932
OSS 8	164.5543 (tlogist)	220.2266	204.8067	262.9434 (bi)	263.2098 (bi)	161.8313 (Ep)	169.5369 (Ep)	262.1010	262.2544	157.0412	167.9424

Table 10
Summary of predictive performance in terms of N_{best} and N_{better} across 16 datasets.

Category	Model ID	N_{best} (t)			N_{better} (t)		
		Early	Middle	Late	Early	Middle	Late
Baseline	NHPP	0	2	5	–	–	–
	Wavelet	4	2	2	11	12	6
	LSTM	2	3	4	11	8	5
Single-Kernel	SKNW-F	0	0	0	3	6	1
	SKNW-D	0	1	0	3	6	1
	SKLL-F	1	3	1	13	11	7
	SKLL-D	1	0	1	13	9	5
Multi-Kernel	MKNW-F	2	1	0	3	6	1
	MKNW-D	0	0	0	3	6	1
	MKLL-F	1	2	1	14	10	8
	MKLL-D	5	2	2	14	9	6

Note: N_{better} is calculated relative to NHPP, the traditional baseline method, for each dataset. A method is counted as better if its PMAE is strictly smaller than that of NHPP. For N_{best} , tied best results are counted for all tied models. Bold values indicate the highest count within each prediction phase.

of the fault-growth process has already been observed. Instead, NHPP benefits from suitable parametric assumptions, while LSTM benefits from the increased amount of training information. Kernel-based models remain effective on several datasets, but their relative advantages are more dataset-specific.

Fig. 4 illustrates the late-phase prediction trajectories on OSS 2. After the observation point, the actual cumulative fault-count continues to increase gradually. LSTM follows this growth pattern most closely and achieves the lowest PMAE of 3.1643. Wavelet and NHPP moderately overestimate the subsequent fault growth, whereas SKLL-D and MKLL-D produce substantially steeper trajectories and considerably overestimate the final cumulative fault count. This example confirms that greater model flexibility does not necessarily improve prediction accuracy when the remaining fault-growth process is relatively stable.

5.2. Comparison between NHPP-based SRMs and kernel-based nonparametric models

Although Wavelet and LSTM are included as additional baselines, this subsection focuses on the comparison between NHPP-based SRMs and kernel-based nonparametric models. NHPP represents the classical parametric modeling framework in software reliability evaluation, whereas the kernel-based methods represent a flexible nonparametric alternative. Table 11 presents a model-family-level comparison in which the lowest PMAE among the eight kernel-based methods is compared with the PMAE of NHPP for each dataset and prediction phase.

The kernel-model family achieves a lower PMAE than NHPP on 15, 13, and 9 of the 16 datasets in the early, middle, and late phases,

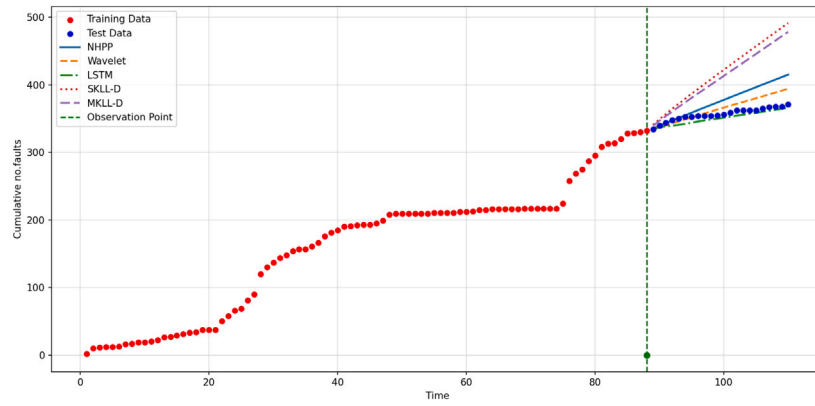


Fig. 4. Prediction trajectories on OSS 2 in the late testing phase using 80% of the observations for training.

Table 11

Model-family-level comparison between NHPP-based SRMs and kernel-based nonparametric models across 16 datasets.

Phase	Kernel Wins	NHPP Wins	Kernel Win Rate
Early (20%)	15	1	93.8%
Middle (50%)	13	3	81.3%
Late (80%)	9	7	56.3%

Note: For each dataset, the lowest PMAE among the eight kernel-based nonparametric methods is compared with that of the AIC-selected fixed-parameter NHPP model. A kernel-family win is counted when the lowest kernel-based PMAE is strictly smaller than the NHPP PMAE. This comparison describes the empirical performance of the kernel-model family and should not be interpreted as the performance of any single kernel-based model. No tied comparisons were observed.

respectively, corresponding to empirical win rates of 93.8%, 81.3%, and 56.3%. The advantage is most pronounced in the early phase and gradually decreases as the proportion of observed data increases. This result indicates that at least one kernel configuration frequently provides a more accurate prediction than the selected NHPP model, particularly under early and middle observation settings.

This stage-dependent pattern is consistent with the different modeling assumptions of the two approaches. NHPP-based SRMs rely on predefined mean value functions that impose structural constraints on their extrapolation behavior. Under the fixed-parameter setting used in this study, the prediction trajectory remains governed by the selected mean value function. When the underlying fault-detection process deviates from this assumed structure, the selected NHPP model may accumulate extrapolation error, especially during long-horizon prediction from limited observations.

In contrast, kernel-based nonparametric models estimate the mean value function directly from the observed data without assuming a fixed global reliability-growth function. This flexibility may be beneficial when the fault-detection dynamics evolve because of changes in testing intensity, debugging efficiency, software features, or operational conditions. Such adaptability may contribute to the stronger model-family-level performance observed in the early and middle phases.

The advantage of the kernel-model family becomes less pronounced in the late phase. As more observations become available and the remaining prediction horizon becomes shorter, the fault-growth patterns of some datasets can be represented effectively by classical NHPP mean value functions. Consequently, NHPP wins on seven datasets in the late phase, compared with only one in the early phase. These results do not imply that kernel-based models uniformly dominate NHPP. Rather, they demonstrate complementary phase-dependent strengths: the kernel-model family is particularly competitive under sparse and long-horizon prediction conditions, whereas NHPP remains effective when its parametric assumptions are well aligned with the observed reliability-growth process.

Because Table 11 selects the best result from multiple kernel configurations, it should be viewed as a model-family-level descriptive analysis. The performance of each individual model, together with comparisons against Wavelet and LSTM, is reported separately in Table 10.

5.3. Impact of the sequential parameter strategy

To address RQ2, we compare two forecasting configurations for each kernel-based estimator: the fixed-parameter strategy (F) and the sequential updating strategy (D). In the fixed strategy, the optimized bandwidths and, for multi-kernel estimators, kernel weights are kept unchanged throughout recursive forecasting. In the sequential strategy, these parameters are updated along the recursively generated forecasting trajectory. Both strategies use the same closed-loop recursive forecasting protocol without incorporating future actual observations.

Table 12 summarizes the win counts of the F and D configurations across the three testing phases. For each model family and phase, the two configurations are compared over 16 datasets using PMAE. Equal PMAE values are recorded as ties and are excluded from the win counts.

In the early phase, the effect of sequential updating differs substantially between the NW and LL estimators. The D configuration performs better for the LL-based estimators, obtaining 10 wins for SKLL and 11 wins for MKLL, compared with four and five wins for their respective F configurations. This result suggests that sequential adaptation of the bandwidths and multi-kernel weights can help local-linear estimators adjust to long forecasting trajectories when only limited observations are available.

The opposite pattern is observed for the NW-based estimators. The F configuration obtains 14 wins for both SKNW and MKNW in the early testing phase, whereas their D configurations obtain only one and two wins, respectively. This result indicates that repeated parameter updating is not uniformly beneficial and may reduce the stability of NW-based recursive predictions under data-sparse conditions.

In the middle phase, the F configuration becomes more competitive for all four model families. It obtains 11, 9, 12, and 10 wins for SKNW, SKLL, MKNW, and MKLL, respectively. The corresponding D configurations obtain only 1, 5, 4, and 6 wins. This pattern suggests that, once half of the fault-growth process has been observed, the parameters estimated at the observation point can provide a relatively stable basis for recursive prediction. Repeated updating may respond to fluctuations in the recursively generated sequence and does not consistently improve prediction accuracy.

Table 12

Effects of the sequential parameter strategy: win counts of fixed-parameter (F) and sequential parameter (D) configurations based on PMAE across 16 datasets.

Testing phase	Model	F wins	D wins	Ties
Early (20%)	SKNW	14	1	1
	SKLL	4	10	2
	MKNW	14	2	0
	MKLL	5	11	0
Middle (50%)	SKNW	11	1	4
	SKLL	9	5	2
	MKNW	12	4	0
	MKLL	10	6	0
Late (80%)	SKNW	4	3	9
	SKLL	5	5	6
	MKNW	9	6	1
	MKLL	7	9	0

Note: For each model family and testing phase, F wins, D wins, and ties are counted over 16 datasets. A tie indicates that the fixed and sequential configurations obtain the same reported PMAE.

The late-phase results are more mixed. SKNW produces nine ties, with four F wins and three D wins, indicating that its two configurations often generate nearly identical PMAE values. SKLL is evenly divided, with five wins for each configuration and six ties. MKNW continues to favor the F configuration, which obtains nine wins compared with six D wins. In contrast, MKLL favors sequential updating, with nine D wins and seven F wins.

Overall, the effectiveness of sequential parameter updating depends on both the estimator and the prediction phase. It is most clearly beneficial for LL-based estimators in the early phase, whereas fixed parameters are consistently more effective for NW-based estimators. In the middle phase, all four model families favor the fixed configuration. In the late phase, the differences become smaller and more dataset-dependent, although MKLL retains a modest advantage from sequential updating. These findings show that sequential updating should be regarded as an estimator- and phase-dependent strategy rather than a universally superior configuration.

5.4. Recommendation

Based on the dataset-wise PMAE results in Tables 7 and the summary statistics in Table 10, we derive stage-specific implications for practical model selection. The results indicate that no single model is uniformly optimal across all prediction phases and datasets.

In the early testing phase (20%), MKLL-D is a favorable choice. It achieves the highest N_{best} , obtaining the lowest PMAE on 5 of the 16 datasets, and records an N_{better} value of 14, tied with MKLL-F for the highest count in this phase. These results indicate that LL-based multi-kernel estimators are particularly effective under sparse observations and long prediction horizons. Nevertheless, Wavelet and LSTM also achieve the lowest PMAE on several datasets. Therefore, MKLL-D should be regarded as a strong early-phase option rather than a universally dominant model.

In the middle testing phase (50%), the results are more diverse. LSTM and SKLL-F each achieve the highest N_{best} , with 3 best-performing cases. Wavelet obtains the highest N_{better} value of 12, followed by SKLL-F with 11. Among the multi-kernel models, MKLL-F provides the strongest overall result, achieving 2 best-performing cases and outperforming NHPP on 10 datasets. Thus, SKLL-F and LSTM are suitable candidates when the primary objective is to maximize the probability of obtaining the lowest PMAE, while Wavelet and MKLL-F provide competitive alternatives with relatively consistent performance.

In the late testing phase (80%), NHPP achieves the highest N_{best} , with 5 best-performing cases, followed by LSTM with 4. This result suggests that parametric modeling and learning-based time-series modeling become more effective when most of the fault-growth process has already been observed. Among the kernel-based methods, MKLL-F obtains the highest N_{better} value of 8, while SKLL-F obtains 7. Therefore,

MKLL-F can still be considered a competitive kernel-based alternative, although the overall advantage of kernel modeling is less pronounced than in the early and middle phases.

Overall, model selection should be adapted to the available observation proportion and the characteristics of the fault-detection process. MKLL-D is recommended as a strong candidate for early-stage prediction. In the middle phase, SKLL-F, LSTM, Wavelet, and MKLL-F provide complementary advantages. In the late phase, NHPP and LSTM should be prioritized, while MKLL-F remains a useful nonparametric alternative. These recommendations describe general empirical tendencies across the evaluated datasets and do not imply that any model is optimal for every software project.

6. Conclusion and future work

This paper investigated long-horizon software reliability prediction using a unified kernel-based nonparametric framework. Under a common closed-loop recursive forecasting protocol, single and multi-kernel estimators with fixed and sequential parameter strategies were compared with NHPP, Wavelet, and LSTM on 16 real-world datasets across three prediction phases. The proposed framework was intended as a flexible complement to parametric NHPP-based SRMs rather than as a general replacement.

The results revealed clear phase-dependent performance. In the early phase, MKLL-D achieved the highest N_{best} , obtaining the lowest PMAE on 5 datasets and outperforming NHPP on 14 of the 16 datasets. At the model-family level, at least one kernel configuration outperformed NHPP on 15 datasets. In the middle phase, LSTM and SKLL-F shared the highest N_{best} , whereas Wavelet achieved the highest N_{better} . In the late phase, NHPP obtained the largest number of best-performing cases, followed by LSTM, while MKLL-F remained the kernel model with the highest N_{better} .

The analysis showed that sequential updating was estimator-dependent and phase-dependent. It favored the LL-based estimators in the early phase, whereas the NW-based estimators strongly favored fixed parameters. Most estimator families tended to favor fixed configurations in the middle phase, while the differences became smaller and more dataset-dependent in the late phase. Overall, no single model dominated all datasets and phases. MKLL-D was a strong choice for early prediction; SKLL-F, LSTM, Wavelet, MKLL-F and MKLL-D provided complementary advantages in the middle phase; and NHPP and LSTM became more competitive in the late phase. These findings highlight the importance of selecting models according to the available observation proportion, estimator structure, and fault-detection characteristics.

Future work should extend the present offline closed-loop recursive setting to online, rolling-origin, or expanding-window scenarios,

Table A.13

Overall computational complexity of the proposed prediction strategies.

Strategy	Single-kernel models	Multi-kernel models
Fixed-parameter strategy	$O(Rn^2 + Tn)$	$O(RJn^2 + TJn)$
Sequential updating strategy	$O((U + 1)Rn^2 + Tn)$	$O((U + 1)RJn^2 + TJn)$

Table B.14

Mean PMAE and standard deviation of multi-kernel models in the early testing phase.

Dataset	MKNW-F	MKNW-D	MKLL-F	MKLL-D
CSS 1	$32.4475 \pm 6.67 \times 10^{-7}$	$32.9487 \pm 6.16 \times 10^{-7}$	$28.0770 \pm 8.95 \times 10^{-5}$	27.1626 $\pm$ 0.0148
CSS 2	9.8581 ± 0.0000	$10.0569 \pm 1.52 \times 10^{-6}$	7.2382 $\pm$ 0.0000	9.6456 ± 0.0042
CSS 3	45.6332 ± 0.0024	48.6405 ± 0.1615	15.1011 ± 0.0004	11.8234 $\pm$ 0.1358
CSS 4	3.3622 ± 0.0000	$3.2911 \pm 1.84 \times 10^{-7}$	0.2662 $\pm$ 0.0000	0.3068 ± 0.0008
CSS 5	10.1262 $\pm$ 1.43 $\times 10^{-6}$	$12.4455 \pm 1.96 \times 10^{-6}$	$34.4389 \pm 1.40 \times 10^{-5}$	33.4714 ± 0.1679
CSS 6	$32.0183 \pm 1.97 \times 10^{-6}$	$32.4857 \pm 1.63 \times 10^{-6}$	$23.0663 \pm 2.32 \times 10^{-6}$	20.9457 $\pm$ 0.0015
CSS 7	130.7993 ± 0.0160	130.8611 ± 0.0034	45.4419 ± 0.5477	41.7495 $\pm$ 0.1699
CSS 8	73.1846 ± 0.0098	73.8230 ± 0.0752	25.2706 ± 0.0039	24.6129 $\pm$ 0.0056
OSS 1	$43.4340 \pm 5.15 \times 10^{-6}$	43.4339 ± 0.0001	20.1036 ± 0.1756	14.0345 $\pm$ 0.6123
OSS 2	$177.8064 \pm 2.39 \times 10^{-5}$	179.4976 ± 0.0017	55.4351 ± 0.0406	46.1683 $\pm$ 0.7739
OSS 3	24.6468 $\pm$ 0.0003	25.1398 ± 0.0234	452.4268 ± 4.4647	530.4164 ± 1.9386
OSS 4	$179.6673 \pm 5.55 \times 10^{-6}$	$179.8958 \pm 2.58 \times 10^{-5}$	138.7527 $\pm$ 0.0059	143.2384 ± 0.1093
OSS 5	$436.1082 \pm 7.06 \times 10^{-6}$	$436.3911 \pm 5.91 \times 10^{-5}$	358.7471 $\pm$ 0.1821	369.9175 ± 0.4253
OSS 6	$980.3790 \pm 3.60 \times 10^{-5}$	980.8453 ± 0.0036	699.9824 ± 0.2763	675.1136 $\pm$ 0.2904
OSS 7	$113.5603 \pm 2.93 \times 10^{-5}$	113.8205 ± 0.0005	78.2788 ± 0.1089	64.3596 $\pm$ 0.2230
OSS 8	$344.4777 \pm 6.47 \times 10^{-5}$	$345.9005 \pm 8.24 \times 10^{-5}$	212.8358 ± 0.1539	175.8876 $\pm$ 1.3999

Note: Each entry reports the mean PMAE $\pm$ standard deviation over 20 repeated runs. The smallest mean PMAE among the four multi-kernel variants for each dataset is highlighted in bold.

Table B.15

Mean PMAE and standard deviation of multi-kernel models in the middle testing phase.

Dataset	MKNW-F	MKNW-D	MKLL-F	MKLL-D
CSS 1	1.6616 $\pm$ 8.22 $\times 10^{-7}$	$1.9733 \pm 2.88 \times 10^{-6}$	15.9809 ± 0.0015	17.1097 ± 0.0009
CSS 2	$7.3504 \pm 1.09 \times 10^{-6}$	$7.4336 \pm 3.18 \times 10^{-7}$	$1.3109 \pm 6.83 \times 10^{-7}$	1.0424 $\pm$ 0.0827
CSS 3	10.6246 $\pm$ 1.02 $\times 10^{-5}$	$11.0847 \pm 2.21 \times 10^{-5}$	19.8558 ± 0.0018	22.5577 ± 0.0087
CSS 4	$1.3371 \pm 3.05 \times 10^{-6}$	$1.3922 \pm 9.31 \times 10^{-7}$	0.7237 $\pm$ 0.0120	0.9217 ± 0.0082
CSS 5	$17.6952 \pm 5.32 \times 10^{-6}$	$17.7871 \pm 1.73 \times 10^{-6}$	10.3381 $\pm$ 0.0048	10.4119 ± 0.0025
CSS 6	$6.2226 \pm 1.20 \times 10^{-5}$	5.7533 $\pm$ 0.0001	54.5453 ± 0.1269	63.2618 ± 0.3213
CSS 7	54.9984 ± 0.1343	54.9102 $\pm$ 0.0141	83.2048 ± 0.0763	82.5734 ± 0.0127
CSS 8	57.2752 ± 0.7479	58.4753 ± 0.2606	37.4788 $\pm$ 2.9539	40.0515 ± 0.5252
OSS 1	9.9899 ± 0.0333	9.9763 $\pm$ 0.0315	30.1002 ± 0.0833	39.9439 ± 0.3830
OSS 2	79.7422 ± 0.1031	79.7926 ± 0.0516	23.4027 ± 0.0419	19.9595 $\pm$ 0.5420
OSS 3	4.1689 ± 0.0111	4.1663 $\pm$ 0.0016	10.7960 ± 16.2633	5.3858 ± 1.6006
OSS 4	240.4964 ± 0.0103	240.5553 ± 0.0125	210.6117 $\pm$ 0.0184	210.8133 ± 0.0673
OSS 5	383.6354 ± 0.0606	383.6635 ± 0.1681	76.7107 ± 0.1393	68.1119 $\pm$ 0.2238
OSS 6	503.9031 ± 0.7046	503.9207 ± 0.0901	113.8614 $\pm$ 0.0437	171.4229 ± 0.0304
OSS 7	54.3004 ± 0.3879	54.3837 ± 0.1913	9.7628 $\pm$ 0.0074	12.0411 ± 0.0020
OSS 8	283.6183 ± 0.0437	283.9925 ± 0.0325	143.5829 ± 0.0052	140.0602 $\pm$ 0.0017

Note: Each entry reports the mean PMAE $\pm$ standard deviation over 20 repeated runs. The smallest mean PMAE among the four multi-kernel variants for each dataset is highlighted in bold.

Table B.16

Mean PMAE and standard deviation of multi-kernel models in the late testing phase.

Dataset	MKNW-F	MKNW-D	MKLL-F	MKLL-D
CSS 1	$3.5224 \pm 1.17 \times 10^{-6}$	$3.5295 \pm 1.71 \times 10^{-5}$	0.4858 $\pm$ 0.0075	0.6623 ± 0.0029
CSS 2	$1.5408 \pm 4.28 \times 10^{-7}$	$1.5537 \pm 5.70 \times 10^{-7}$	0.9584 ± 0.0004	0.9573 $\pm$ 0.0006
CSS 3	$2.0915 \pm 1.37 \times 10^{-6}$	2.0903 $\pm$ 2.24 $\times 10^{-6}$	3.6493 ± 0.0549	3.4235 ± 0.0083
CSS 4	$0.5889 \pm 3.32 \times 10^{-6}$	$0.5812 \pm 2.49 \times 10^{-5}$	0.3181 $\pm$ 3.59 $\times 10^{-8}$	$0.3471 \pm 8.56 \times 10^{-7}$
CSS 5	1.9939 $\pm$ 3.88 $\times 10^{-6}$	$2.0052 \pm 1.06 \times 10^{-5}$	8.3023 ± 0.0102	7.7477 ± 0.0043
CSS 6	$2.3333 \pm 7.42 \times 10^{-7}$	$2.3333 \pm 1.06 \times 10^{-5}$	2.2391 ± 0.1059	1.9542 $\pm$ 0.6523
CSS 7	2.7358 ± 0.0263	2.7310 $\pm$ 0.0040	24.6093 ± 0.0037	28.3071 ± 0.2488
CSS 8	19.0955 ± 0.0600	19.1036 ± 0.0104	4.0654 ± 0.0013	2.9596 $\pm$ 0.0002
OSS 1	7.4238 ± 0.0008	7.4234 ± 0.0002	2.1687 ± 0.1095	2.1204 $\pm$ 0.0358
OSS 2	20.4776 $\pm$ 0.0291	20.4920 ± 0.0129	37.7164 ± 0.3676	53.6525 ± 0.4280
OSS 3	0.9699 ± 0.0004	0.9698 $\pm$ 0.0001	1.0766 ± 0.0064	1.0750 ± 0.0028
OSS 4	151.0867 ± 0.0990	151.1005 ± 0.0362	39.3875 $\pm$ 1.5829	48.7939 ± 0.4928
OSS 5	212.9208 ± 0.1424	212.8908 ± 0.0471	98.1103 ± 0.0050	96.6448 $\pm$ 0.0230
OSS 6	231.8644 ± 0.2965	231.8751 ± 0.0877	82.6313 $\pm$ 0.0472	83.4414 ± 0.0135
OSS 7	11.1114 ± 0.0225	11.1177 ± 0.0389	5.4514 ± 0.0010	5.3932 $\pm$ 0.0005
OSS 8	262.1010 ± 2.1337	262.2544 ± 1.2815	157.0412 $\pm$ 2.5648	167.9424 ± 1.2108

Note: Each entry reports the mean PMAE $\pm$ standard deviation over 20 repeated runs. The smallest mean PMAE among the four multi-kernel variants for each dataset is highlighted in bold.

where models are updated only when new actual fault-count observations become available. Another important direction is to develop adaptive or sparsity-aware methods for learning kernel weights and bandwidths, which may reduce computational cost while maintaining prediction accuracy. In addition, uncertainty quantification and monotonicity-constrained kernel estimators should be incorporated to support more reliable comparison and risk-aware decision-making. Applying the proposed framework to larger industrial systems, including continuous integration environments, cloud platforms, and AI services, would further assess its generalizability and practical relevance.

CRedit authorship contribution statement

Jiacheng Bai: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Tadashi Dohi:** Writing – review & editing, Supervision, Methodology. **Jingchi Wu:** Software. **Junjun Zheng:** Supervision. **Hiroyuki Okamura:** Supervision, Resources.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by JST SPRING, Grant Number JPMJSP 2132.

Appendix A. Computational complexity

Table A.13 summarizes the computational complexity of the proposed prediction strategies. Here, n denotes the number of observations in the working dataset, T denotes the prediction horizon, J denotes the number of base kernels in the multi-kernel model, R denotes the number of LSCV evaluations required for parameter selection, and U denotes the number of parameter re-estimation steps during sequential updating.

Appendix B. Repeated-run stability results for multi-kernel models

Tables B.14–B.16 report the mean PMAE and standard deviation over 20 repeated runs. Except for a few individual cases, the standard deviations are generally small relative to the corresponding mean PMAE values, suggesting that the multi-kernel optimization procedure is reasonably stable under the experimental settings used in this study.

Data availability

Research Link Provided

[Software Fault-Detection Data \(Original data\)](#) (Github)

References

- [1] M.R. Lyu (Ed.), *Handbook of Software Reliability Engineering*, McGraw-Hill, New York, 1996.
- [2] A.L. Goel, K. Okumoto, Time-dependent error-detection rate model for software reliability and other performance measures, *IEEE Trans. Reliab.* 28 (1979) 206–211.
- [3] S. Yamada, M. Ohba, S. Osaki, S-shaped reliability growth modeling for software error detection, in: *Proceedings of IEEE International Symposium on Reliability and Maintainability*, IEEE CPS, 1983, pp. 475–484.
- [4] J.D. Musa, A. Iannino, K. Okumoto, *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, New York, 1987.
- [5] Y. Chen, N.D. Singpurwalla, A non-Gaussian Kalman filter model for tracking software reliability, *Statist. Sinica* 4 (1994) 535–548.
- [6] E.A. Nadaraya, On estimating regression, *Theory Probab. Appl.* 9 (1) (1964) 141–142.
- [7] G.S. Watson, Smooth regression analysis, *Sankhya: Indian J. Stat. Ser. A* 26 (4) (1964) 359–372.
- [8] J. Fan, I. Gijbels, Variable bandwidth and local linear regression smoothers, *Ann. Statist.* 20 (4) (1992) 2008–2036.
- [9] H. Pham, *System Software Reliability*, Springer Science & Business Media, 2007.
- [10] J.D. Musa, K. Okumoto, A logarithmic Poisson execution time model for software reliability measurement, in: *Proceedings of the 7th International Conference of Software Engineering, ICSE-1984*, ACM, 1984, pp. 230–238.
- [11] N.D. Singpurwalla, S.P. Wilson, *Statistical Methods in Software Engineering: Reliability and Risk*, Springer Science & Business Media, 2012.
- [12] D.R. Miller, A. Sofer, A nonparametric software-reliability growth model, *IEEE Trans. Reliab.* 40 (1991) 329–337.
- [13] Y. Saito, T. Dohi, Predicting software reliability via completely monotone nonparametric estimator with grouped data, *J. Syst. Softw.* 117 (2016) 296–306.
- [14] A. Gandy, U. Jensen, A non-parametric approach to software reliability, *Appl. Stoch. Models Bus. Ind.* 20 (2004) 3–15.
- [15] Y. Saito, T. Dohi, Software reliability assessment via non-parametric maximum likelihood estimation, *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* (A) E98-A (10) (2015) 2042–2050.
- [16] Y.S. Kim, K.Y. Song, I.H. Chang, Prediction and comparative analysis of software reliability model based on NHPP and deep learning, *Appl. Sci.* 13 (11) (2023) 6730.
- [17] J. Wang, C. Zhang, Software reliability prediction using a deep learning model based on the RNN encoder-decoder, *Reliab. Eng. Syst. Saf.* 170 (2018) 73–82.
- [18] X. Xiao, T. Dohi, Wavelet shrinkage estimation for non-homogeneous Poisson process-based software reliability models, *IEEE Trans. Reliab.* 62 (2013) 211–225.
- [19] J. Wu, T. Dohi, H. Okamura, Long-term software fault prediction with wavelet shrinkage estimation, *J. Syst. Softw.* 216 (2024) 112123.
- [20] M. Barghout, B. Littlewood, A.A. Abdel-Ghaly, A non-parametric order statistics software reliability model, *Softw. Test. Verif. Reliab.* 8 (1998) 113–132.
- [21] Z. Wang, J. Wang, X. Liang, Non-parametric estimation for NHPP software reliability models, *J. Appl. Stat.* 34 (2007) 107–119.
- [22] T. Dohi, J. Zheng, H. Okamura, Data-driven software reliability evaluation under incomplete knowledge on fault count distribution, *Qual. Eng.* 32 (3) (2020) 421–433.
- [23] L. Dharmasena, P. Zeephongsekul, Fitting software reliability growth curves using nonparametric regression methods, *Stat. Methodol.* 7 (2010) 109–120.
- [24] J. Bai, T. Dohi, J. Zheng, H. Okamura, Comparison of kernel-based nonparametric and parametric regression approaches for software reliability prediction, in: *Proceedings of the 6th World Symposium on Software Engineering, WSSE-2024*, ACM, 2024, pp. 32–37.
- [25] M. Zhao, M. Xie, On maximum likelihood estimation for a general non-homogeneous Poisson process, in: *Proceedings of Scandinavian Conference on Statistics*, 1996, pp. 597–607.
- [26] A.A. Abdel-Ghaly, P.Y. Chan, B. Littlewood, Evaluation of competing software reliability predictions, *IEEE Trans. Softw. Eng. SE-12* (1986) 950–967.
- [27] H. Okamura, T. Dohi, S. Osaki, Software reliability growth models with normal failure time distributions, *Reliab. Eng. Syst. Saf.* 116 (2013) 135–141.
- [28] J.A. Achcar, D. Dey, M. Nivethi, A Bayesian approach using nonhomogeneous Poisson processes for software reliability models, in: *Frontiers in Reliability*, World Scientific, Singapore, 1998, pp. 1–18.
- [29] M. Ohba, Inflection S-shaped software reliability growth model, in: *Stochastic Models in Reliability Theory*, Springer, Berlin, 1984, pp. 144–162.
- [30] S. Gokhale, K. Trivedi, Log-logistic software reliability growth model, in: *Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium, HASE-1998*, IEEE CPS, 1998, pp. 34–41.
- [31] K. Ohishi, H. Okamura, T. Dohi, Gompertz software reliability model: estimation algorithm and empirical validation, *J. Syst. Softw.* 82 (2009) 535–543.
- [32] S. Watanabe, On discontinuous additive functionals and Lévy measures of a Markov process, *Jpn. J. Math.: Trans. Abstr.* 34 (1964) 53–70.
- [33] O.O. Aalen, Nonparametric inference for a family of counting processes, *Ann. Statist.* 6 (4) (1978) 701–726.
- [34] J.D. Musa, Software reliability data, *Data Anal. Cent. Softw. Rome Air Dev. Cent. Griffiss AFB* (1979).
- [35] A. Wood, Predicting software reliability, *Computer* 29 (1996) 69–77.
- [36] M.A. Vouk, Using reliability models during testing with non-operational profiles, in: *Proceedings of the 2nd Bellcore/Purdue Workshop on Issues in Software Reliability Estimation*, IEEE CPS, 1992, pp. 103–111.
- [37] H. Okamura, Y. Etani, T. Dohi, Quantifying the effectiveness of testing efforts on software fault detection with a logit software reliability growth model, in: *Proceedings of the 2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, IEEE CPS, 2011, pp. 62–68.

- [38] Redis Team, Redis: Source code and commit history, GitHub Repository, [Online]. Available: <https://github.com/redis/redis>, Data collected from Sep. 2011 to Sep. 2022.
- [39] Wox Launcher, Wox: Source code and commit history, GitHub Repository, [Online]. Available: <https://github.com/Wox-launcher/Wox>, Data collected from Jan. 2014 to Mar. 2023.
- [40] Backbone.js Team, Backbone: Source code and commit history, GitHub Repository, [Online]. Available: <https://github.com/jashkenas/backbone>, Data collected from Oct. 2010 to Jan. 2023.
- [41] Homebrew Project, Homebrew (brew): Source code and commit history, GitHub Repository, [Online]. Available: <https://github.com/Homebrew/brew>, Data collected from Apr. 2016 to Apr. 2023.
- [42] PyTorch Team, PyTorch: Source code and commit history, GitHub Repository, [Online]. Available: <https://github.com/pytorch/pytorch>, Data collected from Sep. 2016 to Mar. 2023.
- [43] Chart.js Team, Chart.js: Source code and commit history, GitHub Repository, [Online]. Available: <https://github.com/chartjs/Chart.js>, Data collected from Mar. 2013 to Mar. 2023.
- [44] DevDocs Team, DevDocs: Source code and commit history, GitHub Repository, [Online]. Available: <https://github.com/freeCodeCamp/devdocs>, Data collected from Nov. 2013 to Mar. 2023.
- [45] RSSHub Team, RSSHub: Source code and commit history, GitHub Repository, [Online]. Available: <https://github.com/DIYgod/RSSHub>, Data collected from Apr. 2018 to Apr. 2023.