Cross-Validation Under Temporal Dependence: Bias, Remedies, and Practical Implications

Filippo Reina 416318 **Dario Liuzzo** 408241 **Leonardo Cartesegna** 408405

Abstract

Nonparametric regression techniques rely heavily on the optimal selection of smoothing parameters. Standard cross-validation (CV) methods, such as Leave-One-Out (LOOCV) or k-fold CV, operate under the assumption of independent errors. When applied to time series or spatial data exhibiting short-range dependence, these "naive" methods underestimate prediction error, leading to systematically over-fitted models. This report investigates the theoretical failure of standard CV in correlated settings and evaluates dependence-aware alternatives: Block CV, Buffered (Leave-$(2l + 1)$-out) CV, and Walk-Forward Validation. We further elaborate on efficient computational strategies for Neighborhood CV (NCV) in penalized spline settings. We compare these methods using Penalized Splines, Kernel Regression, and Gradient Boosted Trees (XGBoost) across synthetic autoregressive processes and real-world financial and environmental datasets. Our findings provide practical guidelines for model selection in the presence of temporal autocorrelation.

1 Introduction

Cross-validation (CV) is a standard tool for estimating predictive risk and selecting model complexity in a statistical learning setting. In nonparametric regression, predictive performance is affected by smoothing or regularization hyperparameters (e.g., the badwidth h in kernel regression or the penalty parameter λ in penalized splines). CV provides a data-driven approach to selecting these hyperparameters by approximating out-of-sample mean squared error (MSE), and is widely used in the model-selection literature (Arlot & Celisse, 2010).

A key limitation is that most standard CV schemes (e.g., randomized K-fold CV, LOOCV) assume independence of the datapoints: validation observations are treated as independent from the training data. Time-dependent data often violate this assumption, and autocorrelated observations are common in many settings. When errors are serially correlated, observations close in time share information, and random shuffling can use highly correlated neighbors in the training and validation sets. As we will show, this results in leakage between training and validation sets, which leads to optimistically biased CV risk estimates, and selection of overly flexible models (undersmoothing) that overfit to the noise rather than the signal. Dependance-aware alternatives model this by enforcing temporal structure in the splits, for example by blocked or buffered validation (Burman et al., 1994; Racine, 2000). For forecasting problems, common in fields such as econometrics, rolling-origin (or walk-forward) cross-validation has substantial benefits over traditional CV methods.

Research Question

This report studies why naive (randomized) CV can fail under temporal dependence, and evaluates CV schemes designed to reduce leakage. Specifically, using synthetic data with autocorrelated errors, we aim to study:

(i) the size of the bias of naive CV as an estimator of test MSE under different dependence strengths and signal complexities;

(ii) which depence-aware CV schemes have the highest stability and reliable hyperparameter selection;

(iii) how these conclusions vary across regression models (kernel regression, penalized splines, and gradient-boosted trees).

We aim to provide insights on the cost-performance tradeoff of these models, to assess whether the computational cost of each CV scheme results in better performance. While LOOCV is statistically appealing for dependent data, it can be computationally impossible to implement for large datasets. Recent developments show how a broad class of neighborhood CV criteria for quadratically penalized models can be computed at a cost comparable to a single model fit, making them usable in practice (Wood, 2024).

Outline

The report starts by formalizing the regression setting with dependent errors, and explain the mechanism by which naive CV underestimates predictive risk. Next, we describe the dependence-aware CV schemes considered and the computational shortcut used to implement neighborhood CV efficiently for penalized splines.

We then present the simulation results across multiple signals and error structures, and report some practical recommendations. We conclude the report by presenting complementary findings on real-world financial and environmental series that show distinct autocorrelation patterns, followed by a brief discussion and conclusion.

2 Theoretical Framework

We consider the non-parametric regression model

$$y_t = f(x_t) + \varepsilon_t, \qquad t = 1, \dots, n, \tag{1}$$

where x_t is an ordered index (time), $f(\cdot)$ is an unknown regression function, and $\{\varepsilon_t\}$ is a mean-zero, second-order stationary error process. In the simulations, we assess out-of-sample predictive performance under an *independent replicate* of the data generating mechanism: after selecting a hyperparameter θ by cross-validation, we refit the model on the full training series and evaluate mean squared error on a fresh test series generated with the same f but a new realization of $\{\varepsilon_t\}$.

To compare validation schemes, it is convenient to define a generic cross-validated risk estimator. Let $V_k \subset \{1, \dots, n\}$ denote the validation index set for split k and T_k the corresponding training index set. Given a model class indexed by a tuning parameter θ (e.g., bandwidth h, spline penalty λ), let $\hat{f}_{\theta,T_k}$ denote the estimator trained using observations with indices in T_k. The cross-validation estimate of prediction error is

$$\widehat{R}_{\mathrm{CV}}(\theta) = \frac{1}{\sum_{k=1}^{K} |V_k|} \sum_{k=1}^{K} \sum_{t \in V_k} \left(y_t - \hat{f}_{\theta,T_k}(x_t) \right)^2, \tag{2}$$

and the selected hyperparameter is $\hat{\theta}_{\mathrm{CV}} = \arg\min_\theta \widehat{R}_{\mathrm{CV}}(\theta)$.

Covariance Structure and Autocorrelation

Temporal dependence is characterized by the autocovariance function (ACVF)

$$\gamma(h) = \mathrm{Cov}(\varepsilon_t, \varepsilon_{t+h}) = \mathbb{E}[\varepsilon_t \varepsilon_{t+h}], \tag{3}$$

and the autocorrelation function (ACF) $\rho(h) = \gamma(h)/\gamma(0)$. The magnitude and decay of $\rho(h)$ determine how much "information leakage" occurs between training and validation sets. We distinguish four common dependence structures:

1. **Autoregressive processes (AR(p)).** For AR(1), $\varepsilon_t = \phi\varepsilon_{t-1} + \eta_t$ with i.i.d. innovations η_t, the autocorrelation decays geometrically:
$$\rho(h) = \phi^{|h|}.$$
2. **Moving-average processes (MA(q)).** For MA(q), $\varepsilon_t = \eta_t + \sum_{j=1}^{q} \theta_j \eta_{t-j}$, the ACVF has finite support:
$$\gamma(h) = 0 \quad \text{for } |h| > q.$$
3. **ARIMA(p, d, q).** ARIMA models allow integration ($d > 0$) and thus can exhibit long-range dependence in levels. This typically complicates the choice of a "safe" separation between training and validation sets, motivating data-driven choices based on the ACF of appropriately differenced series.
4. **Seasonal dependence.** Seasonal AR/ARIMA models with period s generate autocorrelation spikes at multiples of s:
$$|\rho(s)|,\ |\rho(2s)|,\ \dots \text{ may be large,}$$
implying that any leakage-mitigation strategy should account for the seasonal period.

The Failure of Naive Random k-fold Cross-Validation

We consider standard randomized K-fold Cross-Validation. The index set $\{1, \dots, n\}$ is randomly partitioned into K disjoint folds $V_1, \dots, V_K$. For each fold k, we train on

$$T_k = \{1, \dots, n\} \setminus V_k$$

and validate on V_k. Given a tuning parameter θ (e.g., $\theta = \lambda$ for penalized splines), let $\hat{f}_{\theta,T_k}$ denote the estimator trained using the observations in T_k. The resulting CV score is

$$\widehat{R}_{\mathrm{CV}}(\theta) = \frac{1}{n} \sum_{k=1}^{K} \sum_{t \in V_k} \left(y_t - \hat{f}_{\theta,T_k}(x_t) \right)^2. \tag{4}$$

Claim 1 (Bias of Random k-fold CV)**.** *When data are serially correlated (e.g., $AR(1)$ errors), random shuffling destroys the temporal order but preserves the dependence between an observation and its neighbors.*

In particular, because random shuffling makes it highly likely that near-neighbors of a validation index $t \in V_k$ remain in the training set T_k, the fitted value $\hat{f}_{\theta,T_k}(x_t)$ becomes positively correlated with the contemporaneous error ε_t, which reduces the expected validation loss and can favor overly flexible hyperparameters: $\mathbb{E}\left[\widehat{R}_{\mathrm{CV}}(\theta)\right] \ll \mathbb{E}\left[\textit{True Prediction Error}\right]$.

Proof: negative bias induced by leakage

Fix a split k and a validation index $t \in V_k$. Using the model $y_t = f(x_t) + \varepsilon_t$ and expanding the square,

$$\begin{aligned}\mathbb{E}\Big[\Big(y_t - \hat{f}_{\theta,T_k}(x_t)\Big)^2\Big] &= \mathbb{E}\Big[\Big(f(x_t) - \hat{f}_{\theta,T_k}(x_t)\Big)^2\Big] + \mathbb{E}[\varepsilon_t^2] \\ &\quad + 2\,\mathbb{E}\big[(f(x_t) - \hat{f}_{\theta,T_k}(x_t))\,\varepsilon_t\big] \\ &= \mathbb{E}\Big[\Big(f(x_t) - \hat{f}_{\theta,T_k}(x_t)\Big)^2\Big] + \gamma(0) \\ &\quad - 2\,\mathrm{Cov}\Big(\hat{f}_{\theta,T_k}(x_t), \varepsilon_t\Big),\end{aligned}$$

where $\gamma(0) = \mathrm{Var}(\varepsilon_t)$ and we use $\mathbb{E}[\varepsilon_t] = 0$.

The key term is $\mathrm{Cov}(\hat{f}_{\theta,T_k}(x_t), \varepsilon_t)$. Under randomized K-fold splitting, the immediate neighbors $t-1$ and $t+1$ are very likely to lie in the training set T_k (probability $\approx (K-1)/K$). For smoothing estimators, $\hat{f}_{\theta,T_k}(x_t)$ depends strongly on nearby responses $y_{t\pm1}$, and under positive autocorrelation ε_t is positively correlated with $\varepsilon_{t\pm1}$ (hence with $y_{t\pm1}$, up to the contribution of f). Therefore $\hat{f}_{\theta,T_k}(x_t)$ becomes positively correlated with ε_t, implying

$$\mathrm{Cov}\Big(\hat{f}_{\theta,T_k}(x_t), \varepsilon_t\Big) > 0.$$

Because this positive covariance is subtracted, the expected validation loss is reduced, yielding an optimistically biased CV score:

$$\begin{aligned}\mathbb{E}\Big[\widehat{R}_{\mathrm{CV}}(\theta)\Big] \;&\approx\; R(\theta) \;-\; \mathrm{Bias}_{\mathrm{dependence}}(\theta), \\ &\text{with } \mathrm{Bias}_{\mathrm{dependence}}(\theta) > 0,\end{aligned}$$

where $R(\theta)$ denotes the target predictive risk under a time-respecting (or independent-replicate) evaluation protocol. This explains why randomized CV can underestimate test error and select overly flexible θ (undersmoothing). □

Dependence-Aware CV Schemes.

To mitigate leakage, we modify the way (T_k, V_k) is constructed, so that validation points are well separated from training points by time gaps large enough to make their correlation negligible. The different schemes are summarized visually by Figure 1.

Block K-fold CV. Partition $\{1, \cdots, n\}$ into K contiguous blocks $B_1, \cdots, B_K$ of (approximately) equal size and compute (2). Block CV maintains temporal ordering and avoids shuffling-induced leakage, but training and validation blocks still show correlation near the block boundaries, when dependance persists.

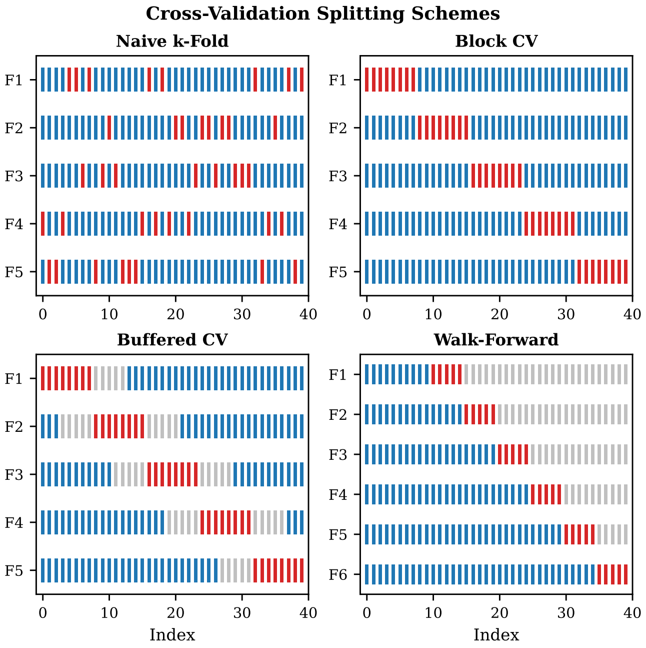

Figure 1: Visual comparison of Cross-Validation schemes. From top left to bottom right: Naive Random K-fold, Block CV, Buffered CV and Walk-Forward Validation. Leave $2\ell + 1$ CV is the same as Buffered CV with one datapoint in the validation set. Red indicates validation sets, blue training sets, and gray the excluded buffer zones.

Buffered Block CV. Buffered block CV removes a buffer zone around each validation block, reducing the correlation that characterizes block CV. Let $\ell \geq 0$ be a buffer radius. If $V_k = B_k = \{a_k, \cdots, b_k\}$, define the buffer indices

$$G_k = \{a_k - \ell, \ldots, a_k - 1\} \;\cup\; \{b_k + 1, \ldots, b_k + \ell\},$$

(truncated to $[1, n]$ at the boundaries), and set

$$T_k = \{1, \ldots, n\} \setminus (V_k \cup G_k).$$

The CV score is computed by (2) using (T_k, V_k). The role of ℓ is to remove lags with large $|\rho(h)|$.

A practical heuristic is to choose ℓ as the smallest lag such that $|\rho(\ell)| \leq \tau$ for a threshold τ (e.g., $\tau = 0.1$) based on an ACF estimate of residuals or a pilot fit. For seasonal dependence, ℓ should be at least the seasonal period.

Leave-$(2\ell + 1)$-out Neighborhood CV (NCV). This is a pointwise variant to Buffered Block CV. NCV leaves out a local neighborhood around each index t:

$$J_t = \{t - \ell, \ldots, t + \ell\},$$

and fits the model on $\{1,\dots,n\} \setminus J_t$. The NCV estimate averages the loss over $t = 1,\dots,n$:

$$\widehat{R}_{\mathrm{NCV}}(\theta) = \frac{1}{n}\sum_{t=1}^{n}\left(y_t - \hat{f}_{\theta,\{1,\dots,n\}\setminus J_t}(x_t)\right)^2.$$

This is statistically well aligned with the leakage mechanism in Claim 1, but naive implementation requires many refits. Section 4 develops an efficient approach for penalized splines that makes NCV feasible inside hyperparameter tuning loops.

Walk-forward (Rolling-Origin) Validation. Walk-forward validation mimics forecasting by ensuring the training set precedes the validation set. Fix an initial training endpoint T_0 and a validation horizon H. For step $j = 0, 1, \dots$ define

$$T_j = \{1,\dots,T_0 + jH\},$$

$$V_j = \{T_0 + jH + 1,\dots,T_0 + (j+1)H\},$$

and compute the average squared loss over all steps. Because it never uses future information, walk-forward validation is appropriate when causality is required.

3 Regression Models and Hyperparameters

This section defines the regression estimators used in our comparisons and clarifies which hyperparameter each cross-validation (CV) scheme selects. Throughout, we treat the predictor as the ordered time index $x_t = t/n \in (0,1]$ and fit models of the form $y_t \approx f(x_t)$. For any training index set $T \subset \{1,\dots,n\}$, we write $\hat{f}_{\theta,T}$ for the fitted regression function under hyperparameter θ trained only on indices in T. In the simulation study, θ is chosen by minimizing a CV estimate of prediction error. Table 1 shows the effects of the hyperparameter choice on the predictions of the regression.

Kernel Regression (Nadaraya–Watson)

Given training data $\{(x_t, y_t)\}_{t\in T}$, the Nadaraya–Watson (Nadaraya, 1964) estimator with bandwidth $h > 0$ is

$$\hat{f}_{h,T}(x) = \frac{\sum_{t\in T} K_h(x - x_t)\, y_t}{\sum_{t\in T} K_h(x - x_t)}, \qquad K_h(u) = \frac{1}{h}K\Big(\frac{u}{h}\Big), \tag{5}$$

where we use a Gaussian kernel $K(u) = \exp(-u^2/2)$. The bandwidth h controls smoothness: small h yields low bias but high variance (potential overfitting), while large h yields smoother estimates.

We tune h over a fixed grid (reported in the Appendix B for reproducibility) and select $\hat{h} = \arg\min_h \widehat{R}_{\mathrm{CV}}(h)$.

Penalized Splines

We approximate $f(\cdot)$ using a p-dimensional B-spline basis (Eilers & Marx, 1996):

$$f(x) \approx \sum_{j=1}^{p}\beta_j B_j(x), \qquad X_{tj} = B_j(x_t),$$

so that fitted values are $X\beta$. For a smoothing parameter $\lambda > 0$, we estimate β by penalized least squares

$$\hat{\beta}_{\lambda,T} = \arg\min_{\beta\in\mathbb{R}^p}\left\{\sum_{t\in T}(y_t - X_t^\top\beta)^2 + \lambda\,\|\Delta\beta\|_2^2\right\}, \tag{6}$$

where Δ is a discrete difference operator (typically second differences), penalizing roughness in the coefficient sequence. The fitted function is $\hat{f}_{\lambda,T}(x) = \sum_{j=1}^{p}\hat{\beta}_{\lambda,T,j}B_j(x)$.

In the Gaussian case, (6) has a closed form:

$$\hat{\beta}_{\lambda,T} = (X_T^\top X_T + \lambda\Delta^\top\Delta)^{-1}X_T^\top y_T, \tag{7}$$

where X_T and y_T denote the design matrix and response vector restricted to indices in T.

Small λ produces flexible fits (risking undersmoothing), while large λ enforces smoothness. Because neighborhood/leave-out validation can require many refits, we use an efficient NCV implementation for penalized splines (Section 4).

We tune λ over a logarithmic grid and select $\hat{\lambda} = \arg\min_\lambda \widehat{R}_{\mathrm{NCV}}(\lambda)$ (see Appendix B).

Gradient-boosted Trees (XGBoost)

We use gradient boosting with decision trees as base learners, as implemented in XGBoost (Chen & Guestrin, 2016). The fitted function is an additive ensemble

$$\hat{f}_T(x) = \sum_{m=1}^{M}\nu\, g_m(x), \tag{8}$$

where each g_m is a regression tree, $\nu \in (0,1]$ is the learning rate, and M is the number of boosting rounds. Trees are fit sequentially to reduce a chosen loss (squared error in our experiments), with regularization to control complexity.

To align with the main question (how dependence affects model complexity selection), we tune the maximum tree depth d, holding other boosting parameters fixed across CV schemes (learning rate, number of rounds, subsampling, etc.). Depth d is a primary driver of variance and overfitting in tree ensembles.

Summary of Tuned Hyperparameters

For comparability across methods, we tune one primary smoothness/complexity parameter per model, presented in Table 1.

Model	Hyperparameter	Effect on complexity
Kernel regression	h (bandwidth)	smaller $h \implies$ more wiggly fit
Penalized splines	λ (penalty)	smaller $\lambda \implies$ less smoothing
XGBoost	d (max depth)	larger $d \implies$ more complex trees

Table 1: Summary of regression models and primary complexity hyperparameters tuned by cross-validation. Smaller h and λ correspond to less smoothing (higher variance), while larger tree depth d increases model flexibility.

4 Efficient Computation for Penalized Splines (NCV)

Neighborhood Cross-Validation (NCV) is statistically attractive for time series regression because it reduces the leakage mechanism identified in Section 2 by omitting a local neighborhood around each validation index. However, a naive implementation is computationally prohibitive: for a series of length n, Leave-$(2\ell + 1)$-out NCV requires fitting the model n times per hyperparameter value, which becomes quickly unfeasible as the dataset size grows.

We overcome this issue by adopting the coefficient approximation method proposed by Wood, 2024.

Penalized Spline Estimator

We estimate the regression function $f(\cdot)$ using a basis expansion. Let $X \in \mathbb{R}^{n\times p}$ be the spline design matrix, $\beta \in \mathbb{R}^p$ the coefficient vector, and $f(x_t) \approx (X\beta)_t$. For a smoothing/penalty hyperparameter λ, the penalized objective can be written as

$$L_\lambda(\beta) = D(y, X\beta) + \frac{1}{2}\beta^\top S_\lambda \beta, \tag{9}$$

where $D(\cdot)$ is a (negative log-likelihood or squared-error) loss and S_λ is the penalty matrix (e.g., for P-splines with a difference operator Δ, one can take $S_\lambda = \lambda\,\Delta^\top\Delta$). In the Gaussian/squared-error case,

$$D(y, X\beta) = \frac{1}{2}\|y - X\beta\|_2^2, \qquad \nabla^2 D(y, X\beta) = X^\top X.$$

Let $\hat{\beta}_\lambda$ denote the minimizer of (9).

Neighborhood CV Objective and Naive Cost

For each index i, define a neighborhood

$$J_i = \{i-\ell, \ldots, i+\ell\} \cap \{1, \ldots, n\}.$$

NCV evaluates prediction of y_i from a model fitted after removing J_i:

$$\widehat{R}_{\mathrm{NCV}}(\lambda) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \widehat{y}_{i,-J_i}(\lambda)\right)^2, \tag{10}$$

$$\widehat{y}_{i,-J_i}(\lambda) = X_i^\top \hat{\beta}_{\lambda,-J_i}, \tag{11}$$

where $X_i^\top$ denotes the ith row of X, and $\hat{\beta}_{\lambda,-J_i}$ minimizes (9) computed on the reduced dataset $\{1, \ldots, n\} \setminus J_i$.

Naive Complexity. If one penalized spline fit costs $O(np^2)$ (dense linear algebra with p basis coefficients), then computing (10) by refitting n times costs $O(n^2p^2)$ per λ, which is infeasible for large n or for tuning over a grid of candidate λ values. The computational shortcut will reduce computational cost to $O(np^2)$, comparable to a single model fit.

FastNCV idea: one Newton Step from the Full-data Solution

Following Wood, 2024, we avoid refitting from scratch by approximating each leave-neighborhood solution $\hat{\beta}_{\lambda,-J_i}$ using a single Newton step starting from the full-data optimum $\hat{\beta}_\lambda$.

Write the full objective as $L_\lambda(\beta)$ and define the contribution of the omitted points as

$$D_{J_i}(\beta) = \sum_{t\in J_i} d_t(\beta),$$

so that the reduced objective is $L_{\lambda,-J_i}(\beta) = L_\lambda(\beta) - D_{J_i}(\beta)$. Let

$$\begin{aligned} H_\lambda &= \nabla^2 L_\lambda(\hat{\beta}_\lambda), \\ g_{J_i} &= \nabla D_{J_i}(\hat{\beta}_\lambda), \\ H_{J_i} &= \nabla^2 D_{J_i}(\hat{\beta}_\lambda). \end{aligned}$$

Since $\nabla L_\lambda(\hat{\beta}_\lambda) = 0$, the reduced gradient at $\hat{\beta}_\lambda$ equals $\nabla L_{\lambda,-J_i}(\hat{\beta}_\lambda) = -g_{J_i}$, and the reduced Hessian is $H_{\lambda,-J_i} \approx H_\lambda - H_{J_i}$. A single Newton step gives the approximation

$$\hat{\beta}_{\lambda,-J_i} \approx \hat{\beta}_\lambda + \left(H_\lambda - H_{J_i}\right)^{-1} g_{J_i}. \tag{12}$$

In the Gaussian case, $H_\lambda = X^\top X + S_\lambda$ and this has an especially simple form:

$$g_{J_i} = X_{J_i}^\top \left(X_{J_i}\hat{\beta}_\lambda - y_{J_i}\right), \qquad H_{J_i} = X_{J_i}^\top X_{J_i},$$

where X_{J_i} and y_{J_i} denote rows/elements restricted to indices in J_i.

Accuracy vs Exactness. Equation (12) is a high-accuracy approximation to the leave-neighborhood solution. In our benchmark, this approximation yields CV curves and selected λ values that are numerically indistinguishable from the naive refitting baseline (Figure 2). If required, the approximation can be refined by additional Newton steps, at increased cost.

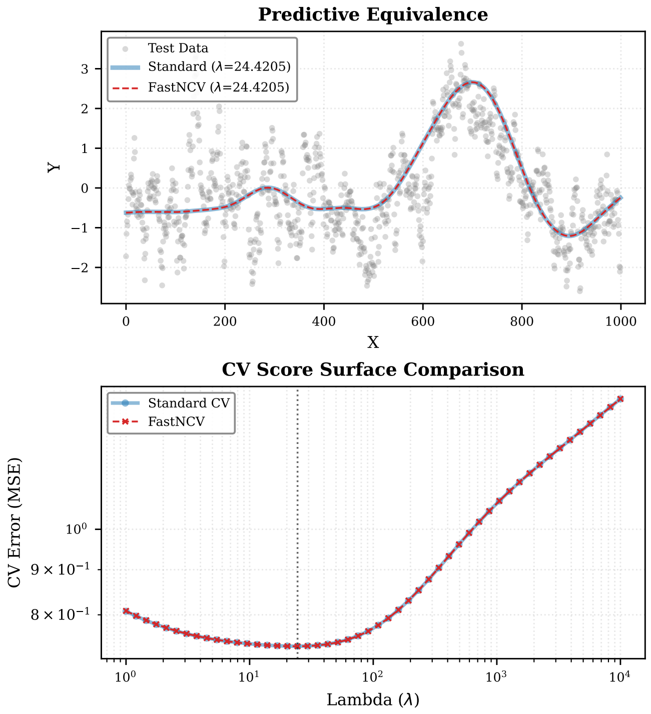

Figure 2: Validation of Numerical Equivalence. Top: The fitted spline functions for Standard CV (thick blue line) and FastNCV (thin red dashed line) perfectly overlap. Bottom: The Cross-Validation error curves are identical, selecting the same minimum.

Cholesky downdating and Woodbury fallback

The computational bottleneck in (12) is solving linear systems with $H_\lambda - H_{J_i}$ for many i. Since $|J_i| = 2\ell + 1$ is small, H_{J_i} is low-rank (rank at most $|J_i|$), so we can update factorizations efficiently.

Compute the Cholesky factor R of the full Hessian:

$$R^\top R = H_\lambda.$$

For each neighborhood J_i, we need to solve

$$\left(H_\lambda - H_{J_i}\right) u_i = g_{J_i}.$$

Rather than recomputing a factorization from scratch, we perform a sequence of rank-one or rank-$|J_i|$ *downdates* to obtain a factor $\tilde{R}_i$ such that $\tilde{R}_i^\top \tilde{R}_i \approx H_\lambda - H_{J_i}$, and then solve using triangular back-substitution. This reduces the per-neighborhood cost to $O(p^2)$ rather than $O(p^3)$.

In rare cases, subtracting H_{J_i} can make the reduced Hessian numerically indefinite (e.g., in sparse regions or for very small λ). When the Cholesky downdate fails, we fall back to a Woodbury-identity based solve for stability.

Algorithm summary

1. Fit the full penalized spline model to obtain $\hat{\beta}_\lambda$ and compute $H_\lambda = \nabla^2 L_\lambda(\hat{\beta}_\lambda)$.
2. Compute the Cholesky factor R such that $R^\top R = H_\lambda$.
3. For $i = 1, \ldots, n$:
   (a) Form g_{J_i} and H_{J_i} from the omitted neighborhood J_i (Gaussian case: $g_{J_i} = X_{J_i}^\top (y_{J_i} - X_{J_i}\hat{\beta}_\lambda)$ and $H_{J_i} = X_{J_i}^\top X_{J_i}$).
   (b) Downdate R to obtain a factorization for $H_\lambda - H_{J_i}$; if downdating fails, use a Woodbury solve.
   (c) Solve $(H_\lambda - H_{J_i})u_i = g_{J_i}$ and set $\hat{\beta}_{\lambda,-J_i} \approx \hat{\beta}_\lambda + u_i$.
   (d) Compute $\widehat{y}_{i,-J_i}(\lambda) = x_i^\top \hat{\beta}_{\lambda,-J_i}$ and accumulate $(y_i - \widehat{y}_{i,-J_i}(\lambda))^2$.
4. Return $\widehat{R}_{\mathrm{NCV}}(\lambda)$ as in (11).

Complexity

For each λ, FastNCV costs one full fit $O(np^2)$ plus n neighborhood updates at $O(p^2)$ each, for a total of $O(np^2)$ leading order. In contrast, naive NCV refits n times, yielding $O(n^2p^2)$ per λ. This reduction makes NCV feasible inside hyperparameter tuning loops and Monte Carlo studies.

Benchmark: Speed and Numerical Agreement

To validate the complexity reduction empirically, we benchmarked standard (refit-based) NCV against FastNCV. We generated synthetic datasets with sample sizes $n \in \{100, 500, 1000, 2000, 3000, 5000, 10000\}$ using a fixed "local bump" signal and autoregressive noise. For each n, we selected λ for a penalized spline model (30 knots, cubic degree) over a grid of 30 candidate values, with buffer radius $\ell = 10$.

Figure 3 reports runtime scaling and speedup: the standard method grows approximately quadratically in n, whereas FastNCV scales approximately linearly, yielding large practical gains (e.g., $\approx 10\times$ at $n = 1000$ and exceeding $40\times$ by $n = 10000$). Figure 2 assesses numerical agreement: across the benchmark, FastNCV and the refit-based NCV produced overlapping CV surfaces and selected the same minimizing λ, and the resulting fitted splines were visually indistinguishable.

Implication for the Main Experiments. These results justify using FastNCV-based buffered/neighborhood validation within our simulation study, where repeated hyperparameter tuning would otherwise dominate runtime.

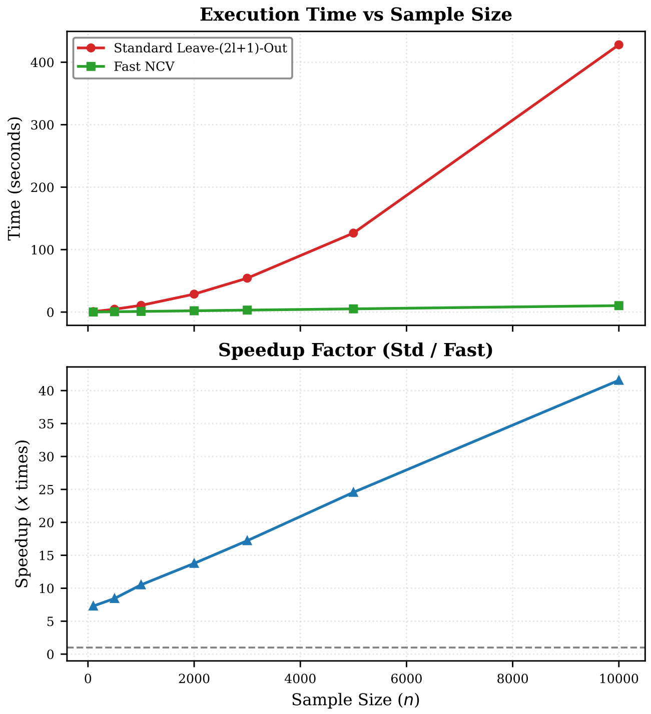

Figure 3: Computational efficiency comparison. Top: Execution time (seconds) vs. Sample Size for Standard NCV (red) and FastNCV (green). Bottom: Speedup factor achieved by the optimized implementation.

5 Simulation Study

We evaluate dependence-aware cross-validation (CV) schemes against a naive randomized baseline using Monte Carlo simulations. The goal is to quantify (i) the bias of CV risk estimation under serial dependence and (ii) the impact on hyperparameter selection and out-of-sample performance.

Each Monte Carlo replication follows the same workflow:

1. Generate a training series $\{(x_t, y_t)\}_{t=1}^{n}$ from a non-parametric regression data generating process (DGP) with dependent errors.
2. For each CV scheme and each model family, select the hyperparameter $\hat{\theta}$ by minimizing the corresponding CV score.
3. Refit the model on the full training series using $\hat{\theta}$.
4. Evaluate performance on the final part of the generated data, which has independent errors from those of the train dataset.

Data generating process (DGP)

We generate data from

$$y_t = f(x_t) + \varepsilon_t, \qquad t = 1, \ldots, n, \tag{13}$$

where x_t is a normalized time index (we set $x_t = t/n$), $f(\cdot)$ is a deterministic signal function, and $\{\varepsilon_t\}$ is a mean-zero dependent error process with $\mathrm{Var}(\varepsilon_t) = \sigma^2$. Unless stated otherwise, we use $n_{\text{train}} = 700$ for training and $n_{\text{test}} = 300$ for testing.

Error Processes. To study the effect of dependence, we consider the following classes (parameters fixed within each experiment):

- **AR(1):** $\varepsilon_t = \rho\, \varepsilon_{t-1} + \eta_t$.
- **MA(1):** $\varepsilon_t = \eta_t + \theta\, \eta_{t-1}$.
- **ARIMA(p, d, q):** an integrated ARMA process (in our runs we use ARIMA$(2, 0, 10)$, i.e. ARMA$(2, 10)$):

$$\varepsilon_t = \phi_1 \varepsilon_{t-1} + \phi_2 \varepsilon_{t-2} + \eta_t + \sum_{j=1}^{10} \theta_j \eta_{t-j}. \tag{14}$$

- **Seasonal AR:** $\varepsilon_t = \rho_1 \varepsilon_{t-1} + \rho_s \varepsilon_{t-s} + \eta_t$.

Here $\{\eta_t\}$ is i.i.d. white noise. Coefficients are chosen to satisfy stationarity/invertibility where relevant.

Signals

We consider three signal families representing different levels of smoothness and localization (Figure 5) (Exact parameter values used in each experiment are reported in Appendix B).

Local Bump. A localized feature implemented via a Gaussian bump:

$$h(x) = a \exp\left\{-\frac{1}{2}\left(\frac{x-c}{w}\right)^2\right\},$$

where a controls amplitude, w controls width, and c sets the location of the bump. To this, we added a baseline

$$f_{\text{sin}}(x) = 0.15 \sin\left(2\pi \frac{x}{l}\right),$$
$$f_{\text{bump}}(x) = h(x) + f_{\text{sin}}(x).$$

Smooth Trend. A smooth trend formed by a low-degree polynomial plus a low-frequency sinusoid:

$$f_{\text{poly}}(x) = \sum_{p=1}^{d} (0.5)^p x^p, \qquad f_{\text{sin}}(x) = A \sin\left(2\pi \frac{x - x_{\min}}{P}\right),$$
$$f_{\text{trend}}(x) = f_{\text{poly}}(x) + f_{\text{sin}}(x).$$

Nonlinear Curvature. A very smooth nonlinear shape intended to mimic gradual growth with a kink:

$$S(x) = \tanh(2.5\, s\, x), \qquad g(x) = \frac{1}{1 + \exp\{-8(x-k)\}},$$
$$B_L(x) = -0.6x^3, \quad B_R(x) = 0.9x^3 + 0.2x,$$
$$B(x) = (1 - g(x))B_L(x) + g(x)B_R(x),$$
$$f_{\text{non-lin}}(x) = 0.7S(x) + 0.6B(x).$$

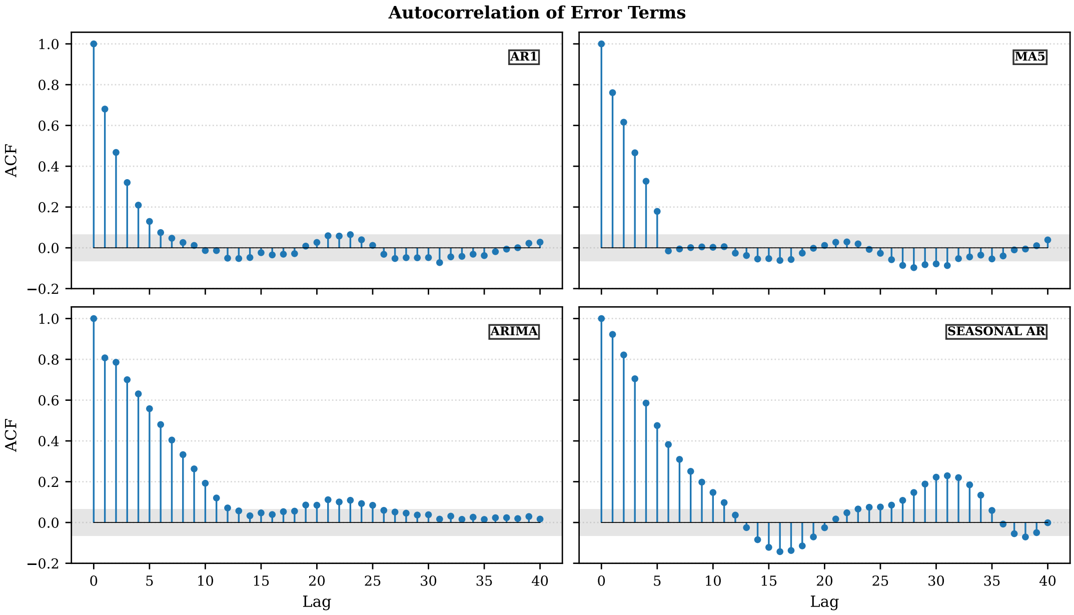

Figure 4: Autocorrelation functions (ACFs) of the simulated *error processes* used in the DGP.

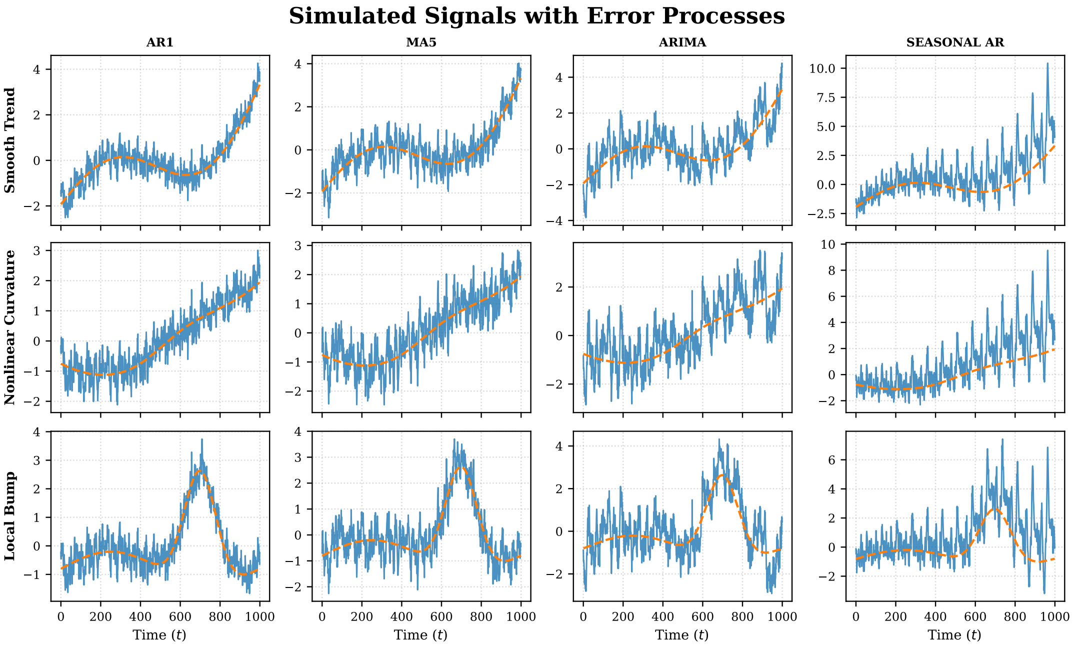

Figure 5: Signal families used in the simulation study (shown with representative noise realizations).

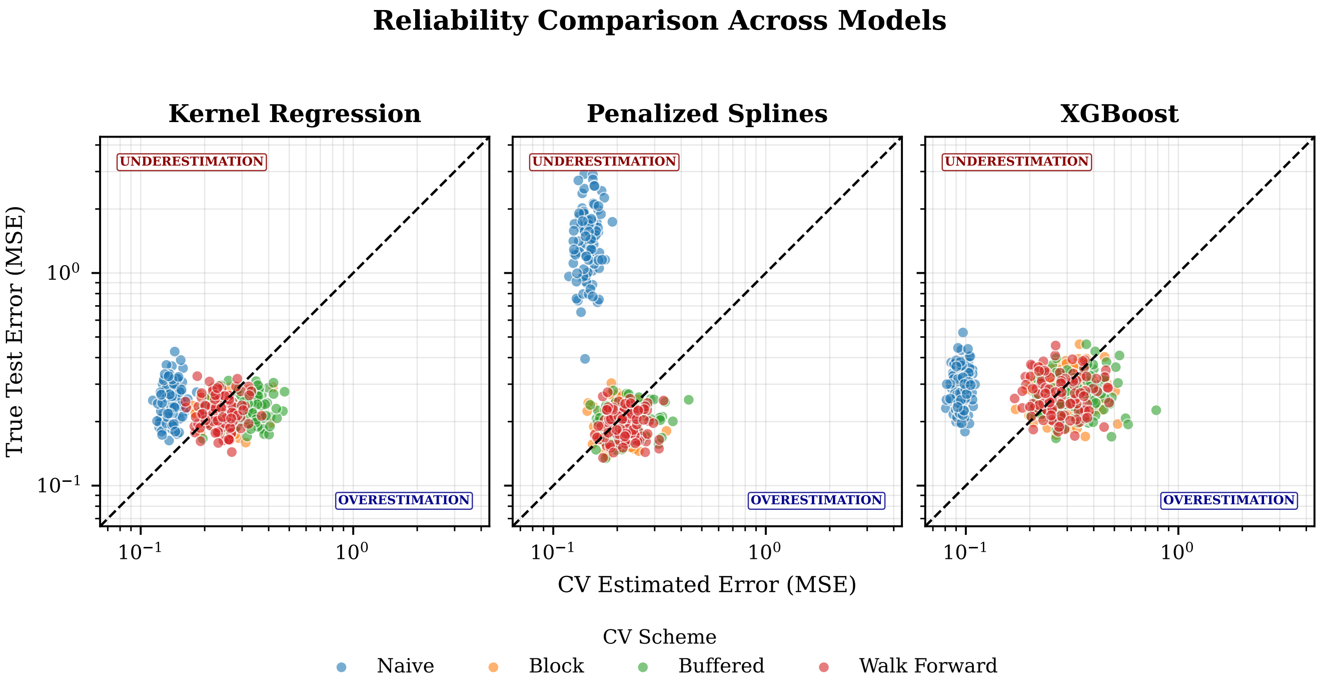

Figure 6: Estimated risk versus independent test risk on synthetic data. Each point corresponds to one Monte Carlo replication, evaluated at the hyperparameter $\hat{\theta}$ selected by the given CV scheme. The diagonal line indicates unbiased estimation. Configuration: signal = non-linear, error process = AR(1), $n = 1000$.

6 Results

This section presents results from the Monte Carlo simulation study. We focus on (i) bias of CV-based risk estimation under temporal dependence, (ii) consequences for hyperparameter selection and fitted function quality, and (iii) sensitivity to signal complexity and dependence structure.

We used some computational shortcuts to avoid having huge runtimes. Specifically, at iteration j we fit the model on the most recent $n_0 = 620$ observations and evaluate multi-step predictive performance over the horizon $\{t_j + 10, \dots, t_j + 20\}$. We then advance the origin and repeat, aggregating errors across all iterations. The final selected $\hat{\theta}$ is used to refit the model and produce the fitted curves shown below.

When fitting splines, instead of the buffered block CV we implement the fast NCV method described above.

Even with this strategy, a complete run of the three regression models on $n = 1000$ datapoints with 100 replications requires between one and two hours of runtime.

Risk Estimation Bias

We first quantify how temporal dependence affects *risk estimation* under different cross-validation (CV) schemes. For each Monte Carlo replication and each CV scheme, we record: (i) the CV-estimated risk at the selected hyperparameter, $\widehat{R}_{\text{CV}}(\hat{\theta})$, and (ii) the independent-test risk (test MSE) of the refitted model, $\text{MSE}_{\text{test}}(\hat{\theta})$. Negative bias indicates optimistic error estimation (underestimation of true predictive error), while positive bias indicates conservative estimation.

Figure 6 plots $\widehat{R}_{\text{CV}}(\hat{\theta})$ against $\text{MSE}_{\text{test}}(\hat{\theta})$ for each replication, with points grouped by CV scheme. The 45° line corresponds to unbiased risk estimation. Points below the diagonal indicate underestimation.

In the configuration shown, that uses an AR(1) error model with the non-linear signal, we distinguish performance of the CV schemes:

- **Naive randomized *K*-fold CV** produces points predominantly *above* the diagonal, indicating optimistic risk estimates. This is consistent with the leakage mechanism in Claim 1: random shuffling leaves temporally adjacent observations in the training set, allowing the fitted model to partially “predict” the correlated noise in the validation fold.
- **Block CV** reduces most leakage; points move closer to the diagonal, though a bias may remain when autocorrelation decays slowly.
- **Buffered block CV** further reduces boundary de-

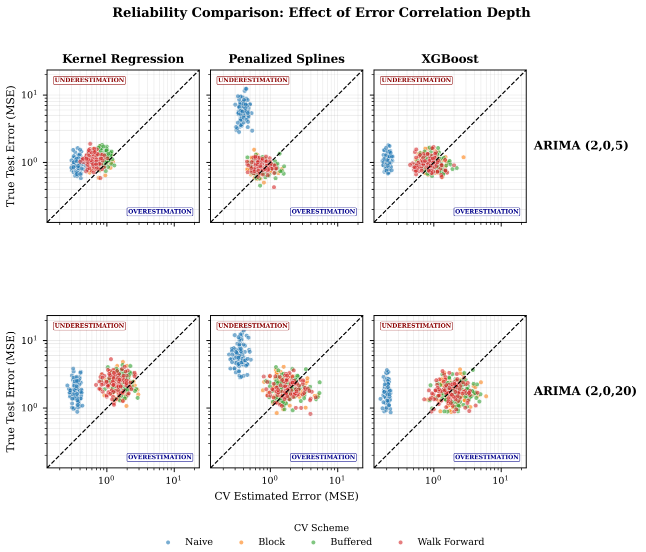

Figure 7: Estimated risk versus independent test risk on synthetic data. Each point corresponds to one Monte Carlo replication, evaluated at the hyperparameter $\hat{\theta}$ selected by the given CV scheme. The diagonal line indicates unbiased estimation. Configuration: signal = non-linear, error process = ARIMA$(2, 0, 5)$ (Top), ARIMA$(2, 0, 20)$ (Bottom), $n = 1000$.

pendence; error estimates become more conservative when ℓ is large, leading to overestimation of the MSE in some cases.

- **Walk-forward validation** produces good results, with points concentrating near the diagonal, indicating accurate risk estimation.

Risk Estimation Bias Across Time-dependence. We compared how the risk estimation bias changes as the autocorrelation decays more slowly. Figure 7 shows that, using an ARIMA(p, d, q), as q increases the bias of the naive CV scheme increases too. However, for that specific DGP (non-linear), the test error for the naive model is sometimes better than that of the time-aware schemes. Moreover, it is easy to notice that the largest impact of time-aware methods is the largest when using penalized splines.

Hyperparameter Selection

Bias in risk estimation directly affects the selected hyperparameter $\hat{\theta}$. Because naive CV systematically underestimates error when neighbors leak information, it tends to select hyperparameters corresponding to *higher model complexity* (undersmoothing/overfitting). In contrast, block-based schemes penalize complexity more strongly and typically select smoother fits.

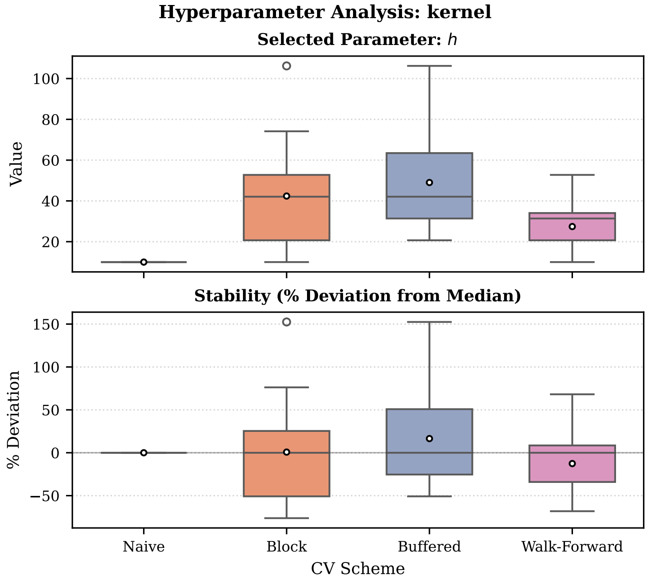

Figure 8: Hyperparameter analysis for kernel regression. Top panel: distribution of the selected bandwidth h across Monte Carlo replications for each cross-validation scheme (Naive, Block, Buffered, Walk-forward); dots indicate the mean and boxes show the interquartile range. Bottom panel: stability of selection, measured as the percent deviation of the selected h from the scheme-specific median (lower magnitude indicates more stable selection). Configuration; signal = non-linear, error process = AR(1).

Figure 8 shows the distribution of selected bandwidths h for kernel regression and a representative model. Naive CV tends to choose smaller h than dependence-aware schemes, consistent with overfitting. It is worth noting that the naive scheme chooses the same hyperparameter almost always, while the deviation of the choices for the other methods is higher.

Comparison of Fitted Curves

Now we compare fitted curves obtained under different CV schemes, focusing on undersmoothing versus oversmoothing as a consequence of the selected hyperparameter. For the walk-forward scheme, we select the hyperparameter $\hat{\theta}$ using rolling-origin validation on the training series.

We choose replications to illustrate the typical behavior observed across runs, using the non-linear signal with MA(5) in Figure 9. Across signals and error structures, we consistently observe the following behaviors:

- **Naive randomized CV (undersmoothing/overfitting).** Fits selected by naive CV tend to track high-frequency fluctuations in the observed series. This

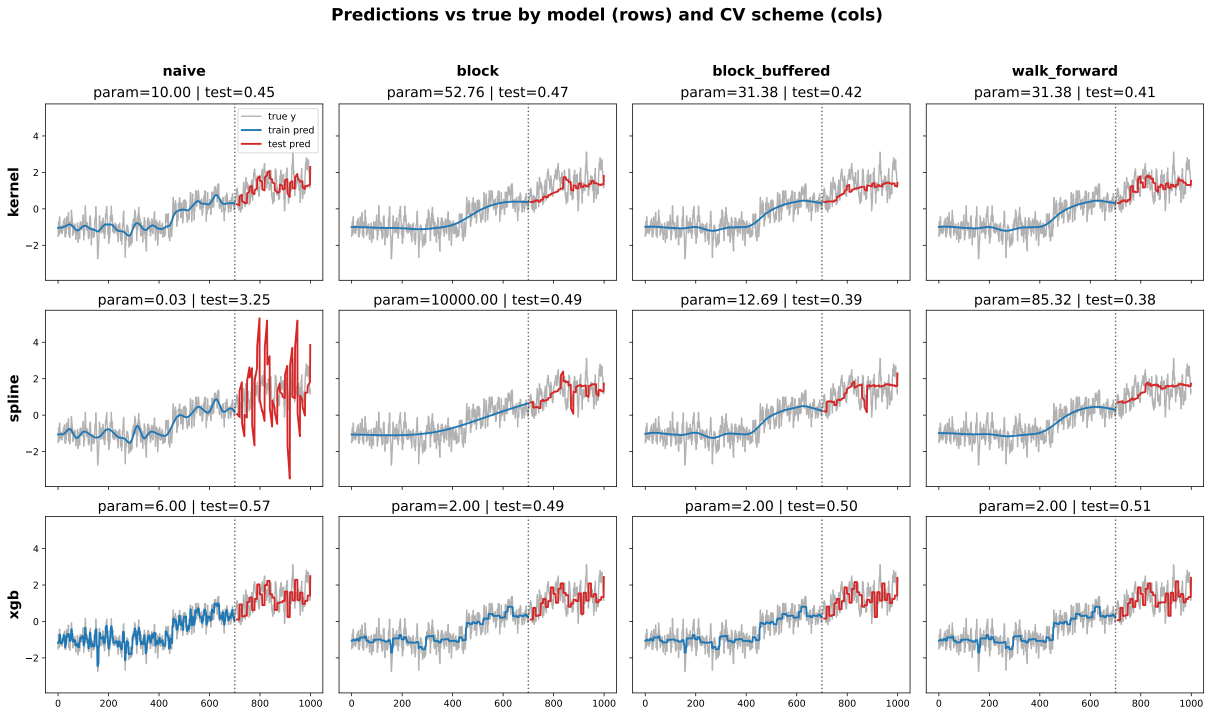

Figure 9: Qualitative comparison of fitted functions under different CV schemes. Configuration: error = MA(5), signal = non-linear.

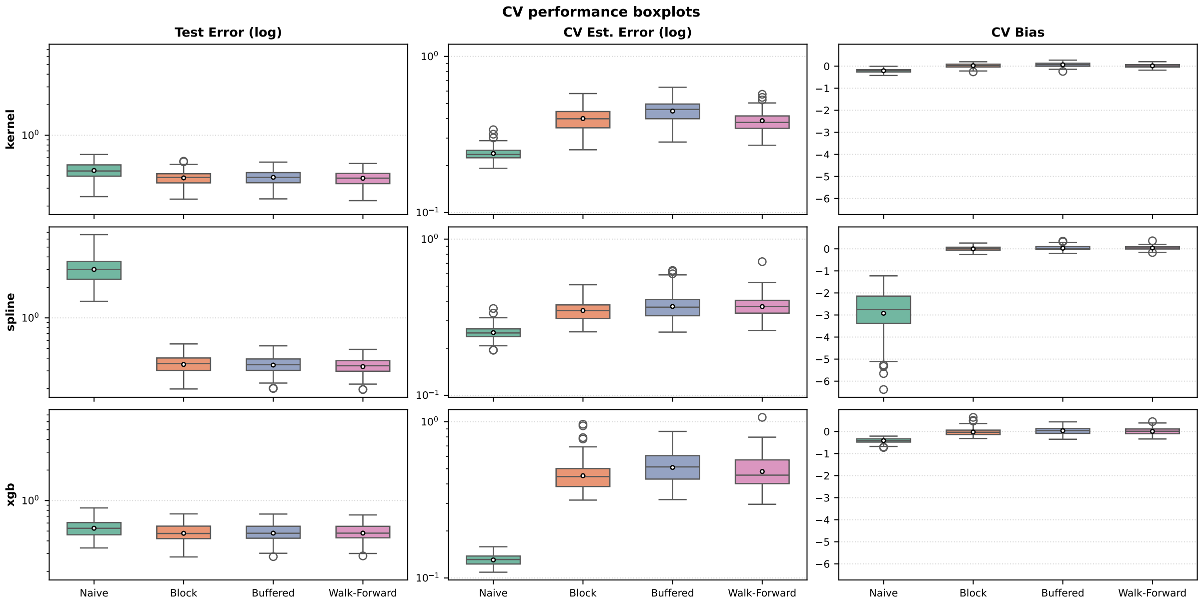

Figure 10: Comparison of CV schemes performance under different regression methods and across 100 independent simulations. Configuration: error = MA(5), signal = non-linear.

behavior reflects leakage: the validation loss is artificially reduced because nearby (correlated) observations remain in the training set. As a result, naive CV frequently selects overly flexible hyperparameters (e.g., small h or small λ), producing wiggly estimates that generalize poorly. This is reflected in higher test MSE in most of the independent 100 simulations, shown in Figure 10.

- **Block CV (reduced leakage, residual boundary effects).** Block CV yields smoother curves than naive CV, especially for persistent processes, but boundary leakage can remain. With non-linear signal and MA(5) error process, the result is satisfying, with test MSE comparable to that of the other methods.
- **Buffered block CV (leakage control, possible conservatism).** Introducing a buffer radius further separates training from validation points and typically leads to smoother fitted functions. For highly persistent or seasonal dependence, this often corrects the undersmoothing seen in the first two methods. However, the performance is very close to Block CV under most scenarios simulated.
- **Walk-forward validation (forecast-aligned fits).** Walk-forward validation produces fits that are typically smoother than naive CV but less conservative than heavily buffered schemes. Since training always precedes validation, walk-forward selection is most useful when applied to time-series forecasting, and adapts quite well to the local structure of the data.

Overall, buffered block CV is the method that reduces leakage the most, and performs slightly better in testing. A more detailed comparison across signals and error generating processes follows in Section 8.

Model Comparison

Overall, splines is the model that is most severely affected by the choice of the hyperparameter. While for kernel density and XGBoost the effect of underestimating the smoothing parameter do not make the predictions unusable, for splines the effects are accentuated. This suggests that, for using splines, time-aware cross-validation is required, while for the other methods the disadvantage of naive CV is mainly the poor result in estimating the MSE.

Time Comparison

The runtime heatmap in Figure 11 indicates that the dominant driver of computational cost is the regression model, not the CV scheme. For the nonlinear signal, kernel regression completes a CV evaluation in roughly 0.02-0.04s across all error types and schemes, while penalized splines are slightly slower at about 0.03-0.06s. By contrast, XGBoost is an order of magnitude more expensive, with times clustered around 0.38-0.47s. Within each model family, the differences between naive, block, buffered, and walk-forward CV are comparatively small, suggesting that choosing a dependence-aware scheme does not impose a substantial additional computational cost relative to the baseline; the notable exception is that walk-forward can be marginally slower for XGBoost due to repeated sequential refits, whereas it can be slightly faster for kernel regression when fewer/smaller training problems are solved per split. This also shows that the fast NCV method for splines is very effective computationally, and has a computational cost similar to the other methods.

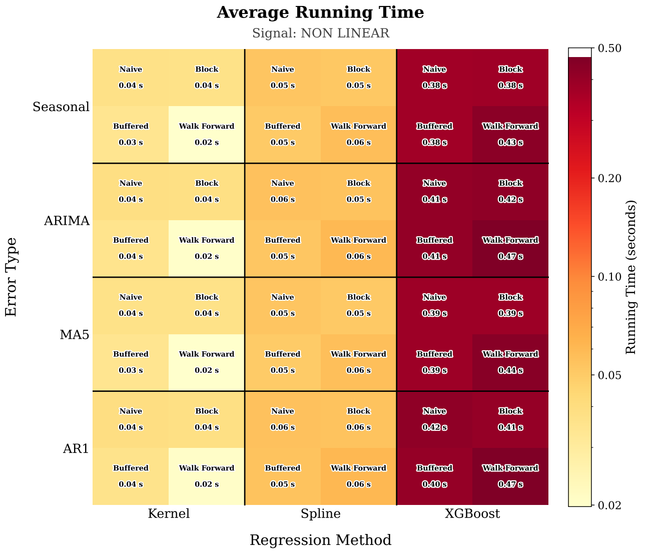

Figure 11: Average running time (seconds; log-scale color map) for cross-validation under the nonlinear signal. Rows correspond to the error type (Seasonal, ARIMA, MA(5), AR(1)); columns correspond to the regression method (Kernel, Spline, XGBoost). Within each panel, the four entries report the mean wall-clock time for Naive randomized K-fold, Block K-fold, Buffered block, and Walk-forward validation. Darker shading indicates longer runtime.

7 Real Data Applications

Following the simulation study, we apply our Cross-Validation frameworks to two real-world datasets. These datasets were specifically selected from the UCI Machine Learning Repository.

For each dataset, we performed stationarity tests using the Augmented Dickey-Fuller (ADF) test (with AIC autolag selection) and inspected the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) to identify the dependence structure.

Dataset Diagnostics and Characterization

Air Quality (Italy). The Air Quality series shows strong short-lag dependence and clear periodic structure: the ACF decays slowly and exhibits pronounced peaks at regular lags (consistent with seasonality). The PACF is dominated by the first lag, suggesting a low-order AR component in addition to the seasonal pattern. This motivates time-respecting validation and, for buffered CV, choosing ℓ at least on the order of the seasonal period (Figure 15).

CNNpred (S&P 500). For close prices, the ACF remains near 1 across many lags and the PACF is dominated by lag 1, consistent with a nonstationary level series. After differencing, both ACF and PACF are close to zero beyond lag 0, indicating an approximately stationary short-memory process. This is representative of an ARIMA process (Figure 16).

Results

Air Quality (Italy). For this dataset, we predicted air quality from 5 to 10 days ahead. Figure 12 shows that randomized CV is overly optimistic on the *Air Quality* series and leads to substantially worse generalization. Naive CV shows much smaller CV error estimates than the time-aware schemes, while producing the largest test errors. Block and buffered block CV inflate the estimated error (particularly for kernel regression), reflecting reduced leakage. These schemes select smoother hyperparameters and achieve significantly lower test MSE. Walk-forward validation performs best overall on this dataset.

For kernel regression, naive CV selects a very small bandwidth ($h \approx 0.2$) and yields a high test MSE, whereas block, buffered, and walk-forward all select a much larger bandwidth and reduce test MSE. For penalized splines, naive CV selects a weak penalty with poor test performance, while the dependence-aware schemes select strong smoothing and achieve test MSE near 1.9. For XGBoost, naive CV selects a deeper tree and performs worse than the time-respecting schemes, which select a shallow tree and improve test MSE to around 2.6. Overall, the real-data behavior closely matches the simulation finding: ignoring temporal structure biases CV toward overly complex models.

CNNpred (S&P 500). Figure 13 compares train/test predictions on the S&P 500 series under hyperparameters selected by each CV scheme. We predict the index level in a window of 20 to 30 days ahead, to simulate forecasting of medium-term returns.

The naive randomized CV scheme selects markedly more aggressive (higher-variance) settings and performs substantially worse out-of-sample than time-respecting schemes. In particular, for kernel regression, naive CV selects a very small bandwidth ($h \approx 6.35$) and yields extremely poor test performance, consistent with severe leakage-driven undersmoothing and instability at the forecast boundary. In particular, the bandwidth chosen is so low that it doesn't match the prediction window, leading to unusable results.

For penalized splines, naive CV likewise selects minimal regularization ($\lambda \approx 0.01$) and exhibits the worst test error, whereas block and walk-forward CV select much stronger smoothing and substantially reduce test error. Buffered block validation produces the lowest test error for the spline model and yields stable forward predictions consistent with the forecasting objective.

For XGBoost, the differences across schemes are smaller than for smoothers and, interestingly, naive CV achieves the lowest test MSE. However, the same qualitative pattern holds: time-respecting schemes avoid the most extreme settings (except the walk-forward CV that selects the same hyperparameter $d = 7$ as the naive one).

8 Discussion

This project studied how temporal dependence compromises standard cross-validation (CV) and evaluated several time-aware alternatives across simulation settings and two real datasets. The results provide a coherent empirical picture that aligns closely with the theoretical leakage mechanism developed in Section 2 (Claim 1): when errors are autocorrelated, random shuffling places highly correlated neighbors of a validation point in the training set, inducing a positive covariance between the fitted value and the contemporaneous noise term, which in turn reduces the CV residual and leads to optimistic risk estimates.

Key Takeaways

Four conclusions emerge.

(i) **Naive randomized CV is not reliable for time series with dependent errors**. In the synthetic study, estimated-versus-test risk plots show that naive CV frequently lies above the 45° line, indicating underestimation of the true out-of-sample error. This optimistic bias translates directly

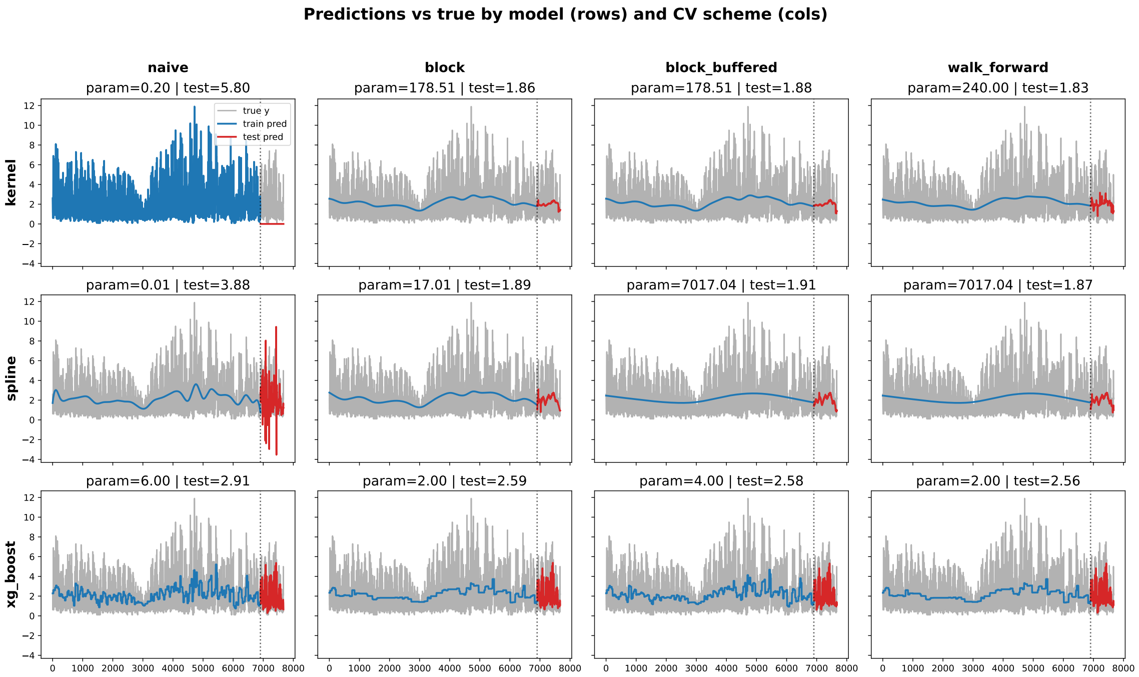

Figure 12: **Air Quality** dataset: train (blue) and test-set predictions (red) under hyperparameters selected by each CV scheme. Rows correspond to regression methods (kernel regression, penalized splines, XGBoost) and columns to CV schemes (naive, block, buffered block, walk-forward). The dashed vertical line marks the train/test split; grey points denote the observed test values. Panel titles report the selected hyperparameter and the resulting test MSE.

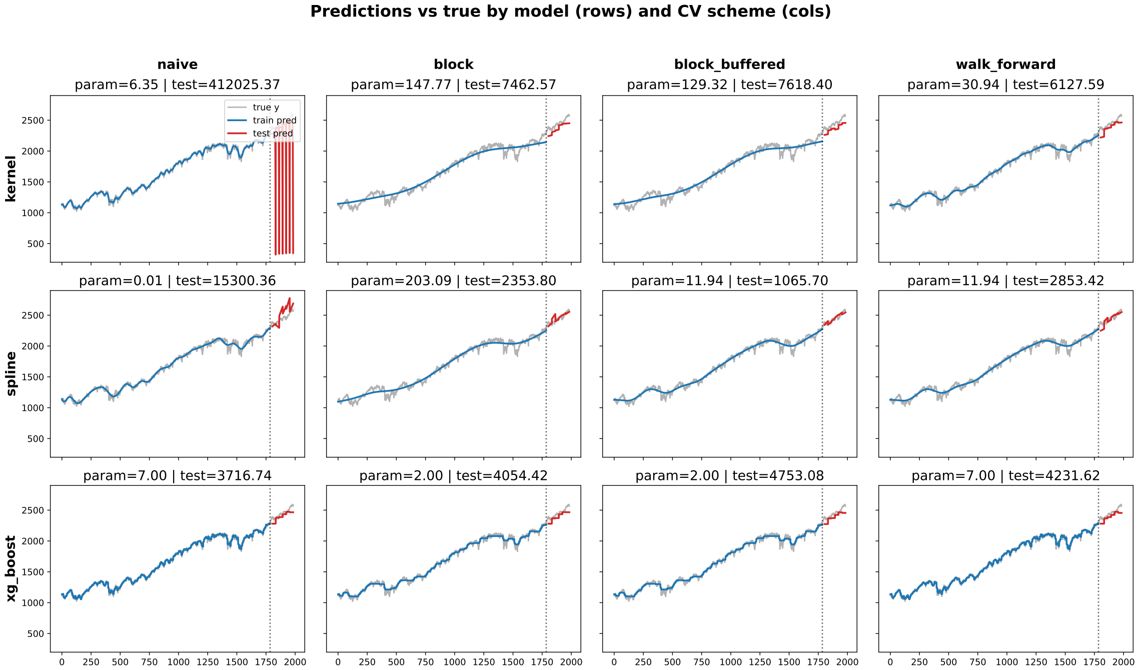

Figure 13: **S&P 500** dataset: train (blue) and test-set predictions (red) under hyperparameters selected by each CV scheme. Rows correspond to regression methods (kernel regression, penalized splines, XGBoost) and columns correspond to CV schemes (naive, block, buffered block, walk-forward). The dashed vertical line marks the train/test split; grey points denote observed test values. Panel titles report the selected hyperparameter and the resulting test MSE.

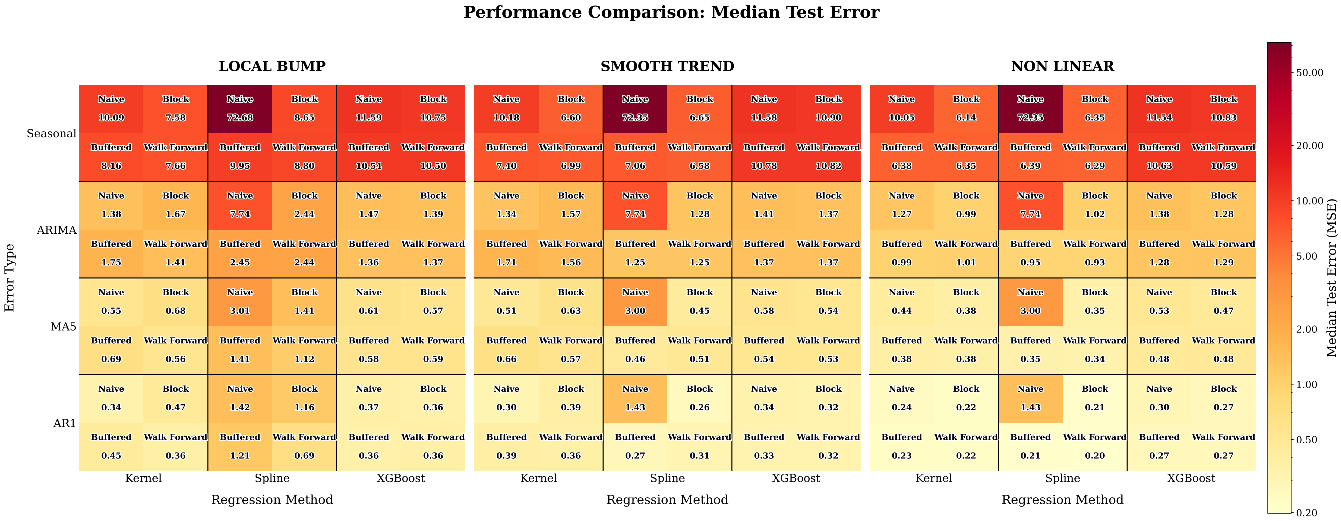

Figure 14: Median test-set error on synthetic data across experimental conditions. Each panel corresponds to a signal type (Local Bump, Smooth Trend, Nonlinear). Rows indicate the error process (Seasonal, ARIMA, MA(5), AR(1)), and within each row the three groups correspond to regression methods (Kernel, Spline, XGBoost). For each method, the four annotated cells report the median test MSE obtained after selecting the hyperparameter using the indicated CV scheme (Naive randomized K-fold, Block K-fold, Buffered block, Walk-forward) and refitting on the full training series. Cell color encodes the median test MSE (darker indicates larger error; color scale shown at right).

into poorer generalization: the median test-MSE heatmap (Figure 14) shows multiple settings where naive CV yields substantially larger test error than time-respecting schemes, especially under persistent dependence and for high-capacity learners.

(ii) **The choice of the regression model impacts the effectiveness of CV**. As discussed when talking about risk estimation bias, Figure 14 shows that the effects of choosing time-unaware CV schemes has the largest impact on splines regression. Thus, while choosing the wrong CV scheme when using KDE or XGBoost leads to underestimation of the test error, the choice of the parameter is still sensible for these models. Using time-aware schemes has multiple advantages for all the regression models, but this choice is even more essential for splines.

(iii) **Time-respecting schemes reduce leakage and stabilize model selection**. Block CV improves over naive CV by preserving ordering, and buffered block CV further reduces boundary leakage by explicitly separating training and validation sets by a lag window. Walk-forward validation typically provides the most realistic error estimate when the evaluation target is forward prediction, since it never trains on observations that occur after the validation period.

(iv) **The impact of CV design depends on both dependence structure and signal complexity**. When the signal is very smooth, many hyperparameters produce similar fits and test errors, so differences between schemes can be modest. However, for localized or harder signals, the bias–variance trade-off becomes sharper: overly optimistic schemes tend to overfit correlated noise, while overly conservative schemes can oversmooth and miss localized structure.

(v) **Computational cost is not a decisive barrier to dependence-aware validation in this setting**. The timing heatmap (Figure 11) indicates that runtime is dominated by the model class (XGBoost $\gg$ splines $\gtrsim$ kernel), whereas differences among CV schemes are comparatively small at the studied scale. This supports using block/buffered schemes and walk-forward for forecasting, without materially increasing runtime.

How the Findings Align with Theory

The empirical findings match the theoretical expectations in two ways. First, the direction of bias is consistent: for positively correlated errors, naive CV is expected to be downward biased because the training fit remains correlated with validation noise. The estimated-versus-test risk plot and the negative CV bias distributions confirm this mechanism. Second, the remedies behave as our theoretical analysis suggested: by enforcing temporal separation between training and validation indices, block and buffered schemes reduce the magnitude of the covariance term driving the bias. Buffered block CV is particularly effective when the ACF decays slowly or exhibits seasonal spikes, because it removes precisely the lags where dependence is strongest.

Practical Recommendations

Based on the combined evidence, we have formulated the following recommendations for choosing the correct CV scheme:

(i) **Avoid randomized CV on time-ordered data.** If the response (or residuals) exhibits autocorrelation, randomized splits will tend to underestimate error and select overly complex models.

(ii) **Use walk-forward validation for forecasting.** When the test set corresponds to future time periods, walk-forward validation has the most realistic error estimates.

(iii) **Choose the buffer size using dependence diagnostics.** A practical approach is to select ℓ based on the empirical ACF (e.g., the smallest lag where $|\hat{\rho}(\ell)|$ drops below a threshold), and to ensure ℓ is at least the seasonal period when seasonal spikes are present.

9 Conclusion

Across both the synthetic experiments and the two real datasets, we find that randomized K-fold cross-validation is systematically unreliable under temporal dependence: it can produce overly optimistic error estimates and consequently select overly complex models (e.g., undersmoothing in kernel/spline regression or excessive complexity in boosted trees). Time-respecting alternatives substantially mitigate this effect. In practice, walk-forward validation is the most appropriate choice for forecasting objectives, while block and buffered/neighborhood-style schemes provide robust model selection for smoothing problems when autocorrelation (or seasonality) is present.

Limitations and Future Work

First, the simulation study focuses on short-to-medium range dependence; long-memory processes would further stress-test buffer design and may require alternative validation strategies. Second, we tune one primary complexity parameter per model family; jointly tuning multiple interacting parameters (particularly for boosted trees) could amplify the consequences of CV bias. Third, real datasets may exhibit structural breaks, time-varying volatility, or missingness mechanisms not captured by the simulation DGP; expanding the empirical study to a broader set of series and explicitly modeling these effects would strengthen generality.

References

Arlot, S., & Celisse, A. (2010). A survey of cross-validation procedures for model selection. *Statistics Surveys*, *4*, 40–79. https://doi.org/10.1214/09-SS054

Burman, P., Chow, E., & NOLAN, D. (1994). A cross-validatory method for dependent data. *Biometrika*, *81*(2), 351–358. https://doi.org/10.1093/biomet/81.2.351

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. https://doi.org/10.1145/2939672.2939785

Chu, C., & Marron, J. (1991). Comparison of two bandwidth selectors with dependent errors. *Ann. Statist.*, *19*. https://doi.org/10.1214/aos/1176348377

Eilers, P. H. C., & Marx, B. D. (1996). Flexible smoothing with B-splines and penalties. *Statistical Science*, *11*(2), 89–121. https://doi.org/10.1214/ss/1038425655

Nadaraya, E. A. (1964). On estimating regression. *Theory of Probability & Its Applications*, *9*(1), 141–142. https://doi.org/10.1137/1109020

Racine, J. (2000). Consistent cross-validatory model-selection for dependent data: Hv-block cross-validation. *Journal of Econometrics*, *99*(1), 39–61. https://doi.org/https://doi.org/10.1016/S0304-4076(00)00030-0

Roberts, D., Bahn, V., Ciuti, S., Boyce, M., Elith, J., Guillera-Arroita, G., Hauenstein, S., Lahoz-Monfort, J., Schröder, B., Thuiller, W., Warton, D., Wintle, B., Hartig, F., & Dormann, C. (2016). Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. *Ecography*, *40*. https://doi.org/10.1111/ecog.02881

Wood, S. N. (2024). On neighbourhood cross validation. *arXiv preprint arXiv:2404.16490*. https://arxiv.org/abs/2404.16490

A Real Data Diagnostics

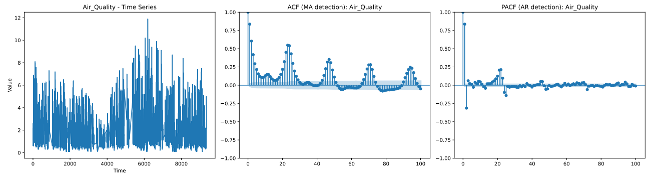

Figure 15: Air Quality (UCI): time series plot (left), ACF (center), and PACF (right) for the response series used in the real-data study. The ACF shows strong short-lag dependence and pronounced repeating peaks at regular lags, indicating seasonal/periodic structure; the PACF shows dominant low-order partial correlations consistent with an autoregressive component in addition to seasonality. Shaded bands indicate approximate 95% confidence intervals.

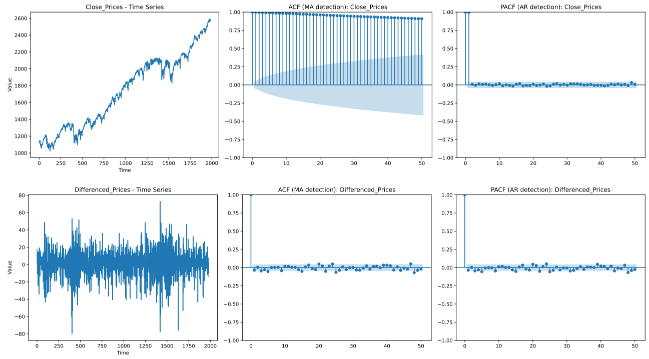

Figure 16: S&P 500: close prices (top row) and first differences (bottom row), with corresponding ACF (center) and PACF (right). The close-price series exhibits extremely persistent autocorrelation consistent with non-stationary levels, while the differenced series shows near-zero autocorrelation beyond lag 0, consistent with a stationary increment/return-like process. Shaded bands indicate approximate 95% confidence intervals.

B Reproducibility and Implementation Details

This appendix documents the exact experimental configuration used in the simulation study, including data-generating process (DGP) parameters, cross-validation (CV) splitting rules, model hyperparameter grids, and software implementation decisions.

Data Generating Process and Experimental Design

The simulation study evaluates model performance using synthetic time series generated via the `synthetic_series.py` and `experiment_helper.py` modules. The data generating process (DGP) follows an additive signal-plus-noise formulation:

$$y_t = f(x_t) + \varepsilon_t, \quad t = 1, \dots, n,$$

where $f(x_t)$ represents the deterministic signal and ε_t represents a stationary, autocorrelated error process. In all primary benchmarks, the sample size is fixed at $n = 1000$.

Deterministic Signal Specifications The covariate grid is defined as an integer sequence $x_t = t \in \{0, \dots, n-1\}$. The signal component $f(x)$ is generated via `signals.py`. To ensure comparability across scenarios with differing functional forms, all signals are standardized (centered to mean 0 and scaled to unit variance) prior to the addition of noise. We examine three distinct signal topologies:

- **Local Bump (Transient Feature).** A localized Gaussian peak designed to test the model's ability to adapt to sudden local changes.

  $$f_{\text{bump}}(x) = \exp\left(-\frac{1}{2}\left(\frac{x-c}{w}\right)^2\right) + 0.15 \sin\left(\frac{2\pi x}{L}\right),$$

  where the center $c \approx 0.7n$, width $w \approx 0.08n$, and L represents the span of the domain.
- **Smooth Trend (Low Frequency).** A cubic polynomial combined with a low-frequency sinusoid, representing smooth, global structural changes without abrupt transitions.

  $$f_{\text{trend}}(x) = \sum_{j=1}^{3} \beta_j x^j + 0.3 \sin\left(\frac{2\pi x}{n}\right).$$

  where $\beta_j = (0.5)^j$.
- **Nonlinear Curvature (Structural Break).** A composite function blending a sigmoid-like transition with cubic bending terms, creating a "soft" structural break.

  $$f_{\text{nonlin}}(z) = 0.7 \tanh(2.5z) + 0.6(0.9z^3 + 0.2z),$$

  where z is the linearly rescaled index $x \in [-1, 1]$.

Error processes and parameters (`utils/errors.py`) All error processes use i.i.d. Gaussian innovations $\eta_t \sim \mathcal{N}(0, \sigma^2)$, where the *innovation* standard deviation σ is passed as a parameter. The code does *not* analytically rescale innovations to impose a fixed marginal variance $\mathrm{Var}(\varepsilon_t)$, and no empirical post-generation rescaling is applied. The marginal variance is therefore implied by the recursion and the chosen σ.

Initialization and burn-in.

- Initial states are zero.
- Burn-in is 200 for AR(1); burn-in is 300 for ARMA/ARIMA and seasonal processes; MA(q) uses padding of q extra innovations but no burn-in.
- Burn-in samples are discarded within `errors.py`.

Processes used in the main experiments.

- AR(1): $\rho = 0.7$, innovation std $\sigma = 0.3$ (notebooks may also test $\rho = 0.8$ in sensitivity runs).
- MA(5): $\theta = 0.7$ with $q = 5$, innovation std $\sigma = 0.3$.
- ARIMA(2,0,10) (i.e. ARMA(2,10)): $(p, d, q) = (2, 0, 10)$ with innovation std $\sigma = 0.45$.
- Seasonal AR: season period $s = 75$, $\rho_1 = 0.4$, $\rho_S = 0.7$, innovation std $\sigma = 0.3$.

Monte Carlo Design and Evaluation Protocol

To ensure the statistical stability of our results, we employ a Monte Carlo design with $R = 100$ replications for each unique scenario. The benchmarks are stratified by signal type, with independent execution notebooks (`local_bump_runs.ipynb`, `non_linear_runs.ipynb`, `smooth_trend_runs.ipynb`) handling the full Cartesian product of error processes and models for their respective signals:

$$\{signal\} \times \{error\} \times \{model\} \times \{CVscheme\},$$

with:

- Signals: Local bump, Nonlinear curvature and Smooth trend (one notebook per signal).
- Errors: AR(1), MA(5), ARIMA(2,0,10), Seasonal AR.
- Models: Kernel regression, Penalized spline, XGBoost.
- CV schemes: Naive k-Fold, Block CV, Block buffered CV, Walk forward.

Replication Loop and Independence. For each replication $r \in \{0, \ldots, 99\}$:

1. **Seed Generation:** Unique random seeds are derived deterministically from the replication index ($S_{\text{train}} = \text{base} + 1000r + 17$). This ensures that the generated error series ε_t are statistically independent across replications while remaining fully reproducible.
2. **Data Synthesis:** The fixed signal $f(x)$ is combined with a fresh realization of the error process ε_t to produce the observed vector y.
3. **Train/Test Split:** The generated series of length $n = 1000$ is partitioned chronologically into a training set ($n_{\text{train}} \approx 700$) and a hold-out test set ($n_{\text{test}} \approx 300$).

Cross-validation implementations (`utils/cv.py`)

- **Shared settings.** Naive and block CV use $K = 5$ folds/blocks by default. Splits are built using `np.array_split`, so fold sizes differ by at most 1 when n is not divisible by K.
- **Naive randomized K-fold.**
  - Shuffle indices once using `default_rng(seed).shuffle`.
  - Split into K folds via `np.array_split`.
  - Each fold is sorted; training is the complement of validation; no stratification.
  - The same shuffled folds are reused for all hyperparameter values within one CV selection run.
- **Block K-fold.**
  - Split ordered indices contiguously via `np.array_split`.
  - Blocks remain chronological.
- **Buffered block CV.**
  - Buffer radius ℓ is a parameter (default 2; main experiments use $\ell = 30$).
  - For validation block $[a, b]$, training excludes $[a - \ell, b + \ell]$ (clipped to $[1, n]$).
  - Buffers are excluded from training only; validation scoring is performed on the validation block itself.
- **Walk-forward validation.**
  - Expanding-window scheme: train on indices $[0, \texttt{train\_end})$, validate on $[\texttt{train\_end}, \texttt{train\_end + val\_size})$, then increment `train_end += step`.
  - Refits at every step; stops when validation would exceed n.

  - In main notebooks: `initial_train_size` $\approx$ 620, `val_size=10` (ahead_h), `step=10` (ahead_l), yielding approximately $\lfloor (1000 - 620)/10 \rfloor = 38$ validation blocks.

Models, tuning grids, and fixed settings

- **Kernel regression (`kernel_regression.py`).**
  - Nadaraya–Watson estimator with Gaussian kernel; a ridge term 10^{-12} is used for numerical stability; no boundary correction.
  - Bandwidth grid in main notebooks: `linspace(10, 320, 30)`.
- **Penalized splines (`penalized_spline.py`).**
  - B-spline basis via `SplineTransformer`, degree 3, `n_knots=30`, `include_bias=True`, linear extrapolation.
  - P-spline penalty uses a difference matrix of order 2: $S = D^\top D$.
  - Lambda grid in main notebooks: `logspace(1e-2, 1e4, 30)`.
  - Solver: linear system $(X^\top X + \lambda S + 10^{-10} I)\beta = X^\top y$ via `scipy.linalg.solve`.
- **XGBoost (`xgb_regression.py`).**
  - Booster: `gbtree`; objective: `reg:squarederror`.
  - Tuned hyperparameter: `max_depth` grid in main notebooks is {2,3,4,5,6,8}.
  - Fixed settings: `n_estimators=200`, `learning_rate=0.05`, `subsample=0.8`, `colsample_bytree=0.8`, `reg_lambda=1.0`, `reg_alpha=0`, `min_child_weight=1.0`, `gamma=0` (default), `n_jobs=1`, `random_state=0`.
  - No early stopping is used.

Notes on timing benchmarks

The `time_comparison` notebook evaluates FastNCV speedups compared to Standard Penalized Splines with Buffered CV (in particular with leave 2ℓ+1 out cross validation) on synthetic data and repeats the computation over `n_values = [100, 500, 1000, 2000, 3000, 5000, 10000]`. Buffer length is set to $\ell = 10$, spline degree is 3, and `n_knots=30`.