informatics

MDPI

Article

Predicting Next-Inspection Cooling-Tower Regulatory Noncompliance to Support Inspection Prioritization: Evidence from New York City

Samer Nofal [1,2]

[1] Department of Computer Science, German Jordanian University, Amman 11180, Jordan; samer.nofal@gju.edu.jo
[2] Department of Computer Science and Engineering, American University of Sharjah, Sharjah 26666, United Arab Emirates

Abstract

Cooling-tower noncompliance can weaken controls intended to limit *Legionella*-related public-health risks. We examined whether longitudinal inspection histories could predict at least one violation at a system's next inspection and support capacity-constrained prioritization. We reconstructed 109,913 retained inspection-result records from the New York City Department of Health and Mental Hygiene, covering January 2017 through March 2026, into 35,825 inspections of 6162 systems. After excluding each system's first inspection, we modeled 29,663 repeat inspections of 5668 systems using 19 history-only predictors. We compared logistic regression, XGBoost, and TabM under system-grouped random and chronological designs, assessed discrimination and probabilistic accuracy, examined model reliance using permutation importance, and retrospectively evaluated prioritization at 10%, 20%, and 30% capacity. The grouped-random test set contained 4217 inspections, of which 1295 were noncompliant; average precision ranged from 0.6695 to 0.6852 and AUROC from 0.7864 to 0.8003. At 10% capacity, the models captured 26.56–27.26% of noncompliant inspections and 42.33–42.64% of violations, with lifts of 2.65–2.72. Performance declined on 5421 chronologically held-out inspections (890 noncompliant): average precision ranged from 0.2645 to 0.2939 and AUROC from 0.6221 to 0.6490. At 10% capacity, and the models captured 18.76–22.02% of noncompliant inspections and 24.05–26.23% of violations, with lifts of 1.87–2.20. The models generally only modestly outperformed four simple history-based rankings. Permutation importance indicated strongest reliance on prior violation burden, inspection-history depth, and elapsed-time measures. Historical records supported retrospective prioritization, but performance weakened for later inspections. Prospective evaluation is needed before operational use; predictions concern recorded regulatory noncompliance, not *Legionella* contamination or direct public-health risk.

Keywords: cooling towers; regulatory violation; next-inspection prediction; inspection prioritization

Check for updates

Academic Editor: Jiang Bian

Received: 18 August 2026
Revised: 9 October 2026
Accepted: 20 October 2026
Published: 23 October 2026

Copyright: © 2026 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Cooling towers remove heat from large air-conditioning, refrigeration, and industrial systems by circulating water and releasing part of it through evaporation. Warm recirculating water and wet internal surfaces can support the growth of *Legionella*, while cooling-tower operation can release fine water droplets that disperse the bacteria into the surrounding air when owners do not adequately operate, clean, disinfect, and monitor the

systems. People may develop Legionnaires' disease after inhaling aerosols that contain the pathogenic *Legionella*. Consequently, failures in cooling-tower maintenance and water management can create public-health risks that extend beyond the buildings in which the systems operate [1–3].

The 2015 South Bronx outbreak demonstrated the potential severity of this risk in a dense urban environment. Investigators linked 138 cases of Legionnaires' disease and 16 deaths to a cooling tower at a South Bronx hotel in New York, USA [2]. New York City responded by enacting Local Law 77 of 2015 and establishing a comprehensive regulatory framework for cooling-tower registration, operation, maintenance, testing, and inspection [4]. The framework requires building owners to register cooling-tower systems, maintain a documented maintenance program and plan, monitor water quality, conduct required testing, perform cleaning and disinfection, retain operational records, and implement corrective actions when necessary. The New York City Department of Health and Mental Hygiene also conducts regulatory inspections that examine maintenance plans, operational records, water-treatment practices, testing activities, and compliance with applicable requirements [4].

A comprehensive regulatory framework does not eliminate the need to determine how inspectors should allocate their attention across a large and heterogeneous population of cooling-tower systems. Routine schedules provide broad coverage, while follow-up and investigative inspections respond to specific concerns. However, systems may differ substantially in their likelihood of future noncompliance. Some systems may consistently satisfy regulatory requirements, whereas others may repeatedly exhibit deficiencies in maintenance, monitoring, documentation, testing, or corrective action. Historical inspection records may therefore contain information that helps regulators identify systems that warrant earlier or more intensive attention.

Noncompliance-informed oversight can use predicted noncompliance as a screening signal while retaining established inspection requirements and professional judgment. In this setting, a prediction model estimates the probability that inspectors will identify at least one regulatory violation during a cooling-tower system's next inspection. Subject to prospective evaluation, regulators could use the resulting rankings to identify systems that may warrant earlier attention or to allocate limited discretionary follow-up capacity. Under a fixed number of available inspection slots, we can assess whether model-based ranking concentrates more noncompliant inspections and recorded violations within the highest-ranked subset than random ordering and simple history-based prioritization rules.

Existing cooling-tower studies have addressed related but distinct problems. Researchers have examined operational and maintenance factors associated with *Legionella* colonization, used microbiological and physicochemical water-quality measurements to estimate colonization, and identified cooling towers in aerial imagery to support outbreak investigations [1,3,5]. Researchers have also used prediction models to prioritize food-establishment and housing-code inspections [6,7]. However, these studies do not establish whether longitudinal cooling-tower inspection histories can predict regulatory noncompliance at the next inspection. Their findings also do not transfer automatically to cooling-tower oversight because regulatory definitions, inspection procedures, outcome frequencies, temporal structures, available predictors, and the consequences of missed noncompliance differ across domains.

The longitudinal and violation-level structure of the NYC cooling-tower data creates specific methodological challenges. The source file can represent a single inspection across multiple violation rows, while each cooling-tower system can undergo multiple inspections over time [8]. We must therefore reconstruct the inspection-level sequence, preserve chronological order, and calculate every historical predictor using only information available

before the target inspection. A random record-level split could place inspections from the same system in both development and test data and mix observations from different periods. To address system-level dependence and temporal change, we use two complementary evaluation designs. The grouped-random system-level design keeps each system entirely within either the development or test partition, whereas the chronological design evaluates models developed from earlier inspections on inspections conducted during later periods. Evaluation procedures for temporally and hierarchically structured data should reflect both the data structure and the intended prediction setting [9].

Transparent evaluation requires more than one summary measure. We assess discrimination using average precision and the area under the receiver operating characteristic curve, and we assess the overall accuracy of the predicted probabilities using the Brier score and logarithmic loss. We report these measures under both grouped-random system-level and chronological evaluation. We also use test-set permutation importance to quantify the extent to which each fitted model's average precision depends on the information in each historical predictor. These reporting and evaluation choices follow established recommendations for prediction-model studies that use regression or machine-learning methods [10].

Accordingly, we model *next-inspection regulatory noncompliance*, not *Legionella* contamination, outbreak occurrence, disease incidence, or individual exposure. The outcome indicates whether inspectors record at least one regulatory violation during the target inspection. It does not directly measure microbiological contamination or public-health harm.

To address the identified gap, we use the longitudinal *NYC Cooling Tower System Inspection Results* data managed by the New York City Department of Health and Mental Hygiene [8]. We reconstruct inspection-level histories, derive predictors exclusively from information available before each target inspection, and use logistic regression, XGBoost, and TabM to predict noncompliance at the next inspection. We evaluate the models under grouped-random system-level and chronological designs and translate their predicted probabilities into inspection-prioritization scenarios. At 10%, 20%, and 30% inspection capacity, we compare the model-based rankings with random ordering and four simple history-based prioritization rules.

We address the following research questions:

1. Regulatory noncompliance: How much predictive information does a cooling-tower system's previous inspection history provide about noncompliance at its next inspection?
2. Predictive performance: How well do logistic regression, XGBoost, and TabM discriminate between compliant and noncompliant next inspections, and how accurate are their predicted probabilities, under grouped-random system-level and chronological evaluation?
3. Inspection prioritization: At 10%, 20%, and 30% inspection capacity, how effectively do model-based rankings identify noncompliant held-out inspections and recorded violations compared with random ordering and four simple history-based prioritization rules?

By answering these questions, we make three main contributions. First, we formulate cooling-tower regulatory noncompliance as a longitudinal next-inspection prediction problem that uses each system's prior inspection history to predict whether inspectors will identify at least one violation at its next inspection. Second, we compare logistic regression, XGBoost, and TabM under grouped-random system-level and chronological evaluation, assess their discrimination and probabilistic accuracy, and compare their reliance on historical predictors. Third, we assess capacity-constrained inspection prioritization by measuring how effectively the model-based rankings identify noncompliant held-out inspections and recorded violations relative to random ordering and four simple history-based rules.

Together, these contributions provide an evidence base for the prospective assessment of noncompliance-informed cooling-tower inspection planning in New York City. Figure 1 summarizes the study design and its principal contributions.

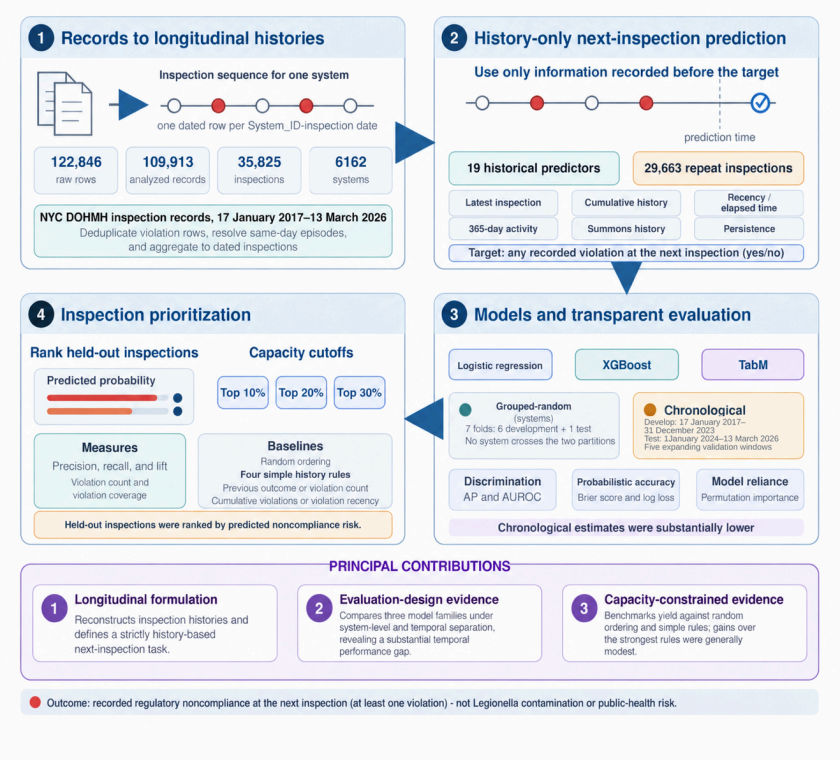


Figure 1. Study design and principal contributions.

2. Related Work

2.1. Cooling-Tower Risk, Compliance, and Predictive Analytics

Studies have linked cooling-tower operation, maintenance, and water management to both *Legionella* colonization and regulatory compliance. Mouchtouri et al. inspected 96 cooling towers in public buildings in Greece and related colonization to deficiencies in risk-management planning, operator training, microbiological testing, disinfection, and chlorine control [1]. Crook et al. analyzed a regulatory intervention covering 1906 sites with evaporative cooling systems in Great Britain. Inspectors identified material legal breaches at 625 sites, with recurring weaknesses in risk assessment, written control schemes, implementation, training, and recordkeeping [11]. These observational studies establish the practical importance of effective control and oversight, but they examine colonization factors or compliance failures already observed during inspection rather than predict a future inspection outcome.

Other studies have used predictive methods for cooling-tower microbiological, environmental, or detection tasks. Campaña et al. fitted logistic-regression models to microbiological and physicochemical water measurements to estimate the probability of *Legionella* colonization in cooling towers and evaporative condensers [3]. In a long-term study of one hospital cooling tower, Girolamini et al. combined microbiological and meteorological measurements with atmospheric-dispersion modeling to examine the possible spread of contaminated aerosols [12]. Wong et al. trained and validated a deep-learning computer-vision model to detect aerially visible cooling towers, thereby supporting source identification during Legionnaires' disease investigations [5]. These studies demonstrate the value of predictive analytics for colonization assessment, dispersion modeling, and tower identification. Their outcomes nevertheless differ from recorded regulatory noncompliance.

This distinction is essential. A cooling tower may violate documentation, monitoring, maintenance, testing, or reporting requirements without direct evidence of *Legionella* contamination. Conversely, an inspection that records no violation does not establish the absence of microbiological risk. The cooling-tower literature therefore motivates effective oversight but does not determine whether a system's longitudinal inspection history can predict whether inspectors will record at least one violation at its next inspection.

2.2. Prediction and Targeting of Regulatory Inspections

Researchers have predicted inspection outcomes in several regulatory domains. Hino et al. used facility and enforcement data to predict failure in water-pollution inspections and evaluated alternative allocations of limited inspection resources [13]. Zhou et al. constructed sliding-window features from enterprise characteristics and compliance records for 16,777 chemical enterprises and compared predicted inspection-failure rates under seven resource-allocation scenarios [14]. Zhou et al. provide a particularly close methodological analog because they used temporal compliance histories to predict future inspection outcomes and evaluated capacity allocation. Neither study, however, addressed cooling-tower regulation or reconstructed violation-level records into repeated inspection-level histories.

Hazardous-waste studies have also examined model-assisted inspection targeting. Manu and Adjei proposed a multi-source framework that combined regulatory, waste-quantity, waste-stream, facility-size, elapsed-time, and toxicity information to predict noncompliance among hazardous-waste generators [15]. They reported projected gains of 46–79% through simulations anchored in prior enforcement benchmarks.

Food-safety inspection studies illustrate both the potential and the evaluation risks of predictive targeting. Kang et al. combined restaurant metadata, previous inspection outcomes and penalty scores, and features from online reviews to distinguish severe violations from inspections with no recorded violation [16]. The study used random cross-validation on an extreme outcome sample rather than chronological next-inspection evaluation. Altenburger and Ho reanalyzed the same data and showed that the apparent advantage of Yelp features depended on the sampling design; on the fuller inspection population, prior inspection history was more informative and Yelp features added little predictive value [17]. This reanalysis directly supports careful outcome sampling, realistic baselines, and temporal evaluation when administrative inspection histories already contain strong predictors.

Other food-safety studies used different evidence sources and evaluation designs. Sadilek et al. developed FINDER from aggregated web-search and location information and used its signals to initiate restaurant inspections in Chicago and Las Vegas. Restaurants identified by FINDER were 3.1 times as likely to be classified as unsafe as restaurants identified through existing methods, providing operational rather than solely retrospective evidence [18]. Oldroyd et al. used business and neighborhood-level characteristics to predict food establishments with unsatisfactory hygiene ratings and proposed prioritizing high-risk outlets awaiting inspection [6].

The City of Chicago's original food-inspection forecasting project provides an important history-based comparator. Schenk et al. combined previous serious or critical violations and time since the previous inspection with business, weather, complaint, neighborhood, and inspector information to predict a critical violation [19]. In a two-month retrospective evaluation, the team reordered the inspections that had already occurred. The model-based order identified 69% of establishments with critical violations in the first half of the period, compared with 55% under the observed order, and identified critical violations about seven days earlier on average. These results quantify a simulated reordering of a fixed test

set; they do not constitute a prospective deployment that changed which establishments were inspected.

Housing and workplace studies broaden the evidence base. Robb et al. used integrated municipal data from proactively inspected properties to predict any housing-code violation, a high-risk health violation, and overcrowding. They estimated that inspecting the highest-ranked properties could yield 1.8 times as many inspections with any violation as existing practice, but this estimate compared observed current yield with model-predicted yield rather than an implemented targeting policy [7]. Johnson et al. combined randomized evidence on the preventive effects of workplace inspections with machine-learning estimates of heterogeneous benefits. Their counterfactual analysis indicated that targeting workplaces with the greatest expected preventive benefit could avert substantially more injuries than existing targeting [20]. This work shows why agencies should evaluate the intended consequence of inspection, not only the probability of recording a violation.

Adjacent engineering and maritime studies have also used machine learning for inspection screening. Rachman and Ratnayake classified oil-and-gas equipment into risk categories for risk-based inspection screening [21], while Tsou predicted ships with high deficiency risk under Port State Control using ship characteristics and previous inspection and detention information [22]. These applications reinforce the usefulness of historical assessments, but their targets represent equipment risk or maritime deficiencies rather than cooling-tower regulatory noncompliance.

2.3. Operational Prioritization, Implementation, and Governance

Predictive accuracy alone does not establish an effective inspection policy. Research on Chicago's food-inspection model illustrates the importance of implementation choices and institutional context. McBride et al. examined the co-creation process and identified the organizational, collaborative, data-infrastructure, communication, and resource conditions that enabled the City to develop the service [23]. Kannan et al. then audited the released data, predictors, model, performance measures, and assumptions, demonstrating why agencies should independently scrutinize predictor definitions and claims about operational gains [24]. Singh et al. showed that differences among inspectors influenced the geographic distribution of time savings and that changes to the scheduling procedure could improve the efficiency–fairness trade-off [25].

The strength of the operational evidence varies across this literature. Several studies estimated targeting performance retrospectively or through simulation [6,7,13–15,19]. FINDER evaluated inspections initiated from model signals [18]. Johnson et al. grounded their policy comparison in randomized evidence about inspection effects, although the alternative targeting rules remained counterfactual [20]. These differences matter because a strong retrospective ranking does not by itself show that a deployed policy will improve inspection yield, timeliness, public health, or equity.

These lessons shape our analysis. We treat predicted probabilities as decision-support information rather than automatic enforcement decisions. We evaluate both system-grouped and chronological generalization, compare model-based rankings with random ordering and transparent history-based rules, and quantify retrospective performance at prespecified inspection capacities. We also use test-set permutation importance to compare how strongly each fitted model's average precision depends on the historical predictors; we do not interpret this analysis as causal or inspection-specific explanation.

2.4. Research Gap and Study Positioning

Prior research has established the public-health importance of cooling-tower control, developed models for microbiological and environmental outcomes, and evaluated risk-

based inspection targeting in several regulatory domains. However, we found no prior study that combines the following elements in the cooling-tower regulatory setting:

1. Reconstruction of longitudinal inspection histories from violation-level regulatory records;
2. History-only prediction of whether inspectors will record at least one violation at a system's next inspection;
3. Grouped-random system-level and chronological evaluation;
4. Comparison of logistic regression, XGBoost, and TabM using discrimination and probabilistic-accuracy measures;
5. Comparison of fitted-model reliance on historical predictors using test-set permutation importance; and
6. Retrospective prioritization at prespecified capacity levels against random ordering and four transparent history-based rules.

We address this gap through a longitudinal next-inspection prediction and evaluation design using New York City regulatory records. We assess whether model-based rankings concentrate noncompliant held-out inspections and recorded violations within the highest-ranked 10%, 20%, and 30% of inspections. This analysis evaluates retrospective ranking performance; it does not estimate the effects of a prospectively implemented inspection policy.

3. Data

3.1. Raw Data Structure

We obtained the data from the *NYC Cooling Tower System Inspection Results* dataset published by the New York City Department of Health and Mental Hygiene [8]. The dataset reports regulatory inspections conducted by the Department's Office of Building Water Systems Oversight. We analyzed a dataset snapshot last updated on 17 April 2026. The snapshot contained posted inspection results with inspection dates from 17 January 2017 through 13 March 2026.

The raw file contained 122,846 rows and 22 fields. It included 6162 distinct cooling-tower systems and 4633 distinct buildings. The fields described system and building identifiers, location, system status and equipment, inspection type and date, recorded violations, health-hazard classifications, citations, and summonses.

The source data used a violation-level structure rather than an inspection-level structure. Inspections with recorded violations could occupy multiple rows, generally one row per recorded violation. The dataset represented an inspection with no recorded violation using a row that retained the inspection information but contained blank violation and citation fields. The raw extract also contained duplicate records, which we address in Section 3.2. Consequently, the 122,846 rows did not represent 122,846 distinct inspections.

The Inspection_Type field classified inspections as *Cycle* or *Non-Cycle*. Cycle inspections were regular compliance inspections, whereas Non-Cycle inspections could include follow-up inspections or other regulatory inspections conducted to evaluate potential public-health risks. Because the dataset did not provide a unique inspection identifier, the raw file contained 35,872 distinct combinations of System_ID, Inspection_date, and Inspection_Type before data cleansing. We did not treat this structural count as the final number of reconstructed inspections.

3.2. Data Cleansing

Before reconstructing the inspection-level records, we examined the raw dataset for exact duplicate rows across all 22 fields. We identified 12,032 duplicate copies beyond the first occurrence among the 122,846 raw records, representing 9.79% of the dataset. For each

group of identical rows, we retained the first occurrence and removed all subsequent copies. This operation left 110,814 records that were unique across all 22 fields. We removed these duplicates before aggregation to prevent repeated copies from inflating the inspection-level violation counts, including the numbers of critical and public-health hazard (PHH) violations.

We next examined the remaining records with a nonmissing Violation_Code for repeated violation entries that were not exact row duplicates. We considered two records to represent the same violation when they had identical values for System_ID, Inspection_date, Inspection_Type, Violation_Code, Law_Section, Violation_Text, Violation_Type, Citation_Text, and Summons_Number. We excluded location, system-status, equipment, and other administrative fields because identical violation entries sometimes contained conflicting values in these fields. For each repeated group, we retained the first occurrence and removed all subsequent copies. This procedure removed 842 repeated violation records and reduced the dataset from 110,814 to 109,972 records.

We identified 47 system-date combinations for which the dataset recorded both a Cycle and a Non-Cycle inspection. These same-day Non-Cycle records may represent follow-up activity associated with the corresponding Cycle inspection. However, the dataset did not provide a unique inspection identifier, inspection time, or follow-up indicator with which to establish the relationship or temporal order between the two records. We therefore applied a consistent episode definition by retaining the Cycle inspection and removing the corresponding 47 Non-Cycle records. Of the removed records, 44 had no recorded violation, whereas each of the remaining three contained one public-health hazard (PHH) violation for failure to take required corrective actions based on bacteriological results. This step reduced the dataset from 109,972 to 109,925 records. We used Cycle as the primary record because it represents the regular compliance inspection. To assess the effect of removing the Non-Cycle records, we conducted a sensitivity analysis that combined information from both inspection types within one episode.

We also identified nine inspection episodes containing both documented violation records and additional records with no value in Violation_Code. These episodes comprised 30 violation records and 11 blank records. Because each blank record shared the same System_ID, Inspection_date, and Inspection_Type as one or more violation records, we treated the blank records as redundant rather than as separate compliant inspections. We removed the 11 blank records, reducing the dataset from 109,925 to 109,914 records. This removal did not change any inspection outcome because each affected episode already contained at least one documented violation.

During a final duplicate check, we identified two records for System_ID 2000000490 on 27 February 2025, with inspection type Non-Cycle, violation code AH8S, and summons number 881321239. The records were identical except that one contained Citation_Text, whereas the other had a missing value in that field. We retained the more complete record and removed the record with the missing citation text. This final step removed one additional repeated violation record and reduced the dataset from 109,914 to 109,913 records.

3.3. Construction of the Inspection-Level Data

After completing the record-level cleaning procedures, we transformed the remaining 109,913 records into an inspection-level dataset. We defined an inspection as a unique combination of System_ID and Inspection_date. Because we had already removed the same-day Non-Cycle records associated with Cycle inspections, each remaining System_ID–Inspection_date combination had only one value of Inspection_Type. We aggregated all records belonging to the same inspection and created one row per inspection.

For each inspection, we set the binary outcome Any_violation to 1 when at least one retained source record had a nonmissing Violation_Code, and to 0 otherwise. We calculated Number_of_violations as the number of retained source records with a nonmissing Violation_Code. We calculated Number_of_PHH_violations and Number_of_critical_violations by counting records classified as public-health hazard (PHH) and Critical, respectively, in Violation_Type. We calculated Number_of_summonses as the number of distinct nonmissing Summons_Number values recorded for the inspection. We also retained Inspection_Type.

The aggregation produced 35,825 inspection-level records representing 6162 cooling-tower systems from 17 January 2017, through 13 March 2026. Of these inspections, 13,933 had at least one recorded violation, corresponding to an overall noncompliance rate of 38.89%. The dataset contained 34,538 Cycle inspections and 1287 Non-Cycle inspections.

Because the predictive models required information from earlier inspections, we examined the amount of recorded history available for each system. Among the 6162 systems, 494 had only one recorded inspection, whereas 5668 had at least two. One record for each system represented its first recorded inspection and therefore had no earlier inspection history. Consequently, 6162 first recorded inspections were ineligible for the primary history-based modeling, leaving 29,663 subsequent inspections from 5668 systems in the modeling cohort.

Figure 2 shows a pronounced temporal change in recorded noncompliance. The annual noncompliance rate decreased from 96.13% in 2017 to 69.85% in 2018 and 35.94% in 2019. It then ranged from 14.30% to 21.28% during 2020–2026. Among the complete calendar years, the annual number of inspections ranged from 2350 in 2025 to 5267 in 2017. The 621 inspections recorded in 2026 cover only the period through March 13 and therefore do not represent a complete annual total. These results indicate that the recorded outcome distribution changed substantially across the observation period.

Because outcome prevalence varied considerably over time, we evaluated the models under both the stratified system-grouped random design described in Section 4.2 and the chronological design described in Section 4.3. We treated the chronological evaluation as the stronger assessment of temporal generalizability.

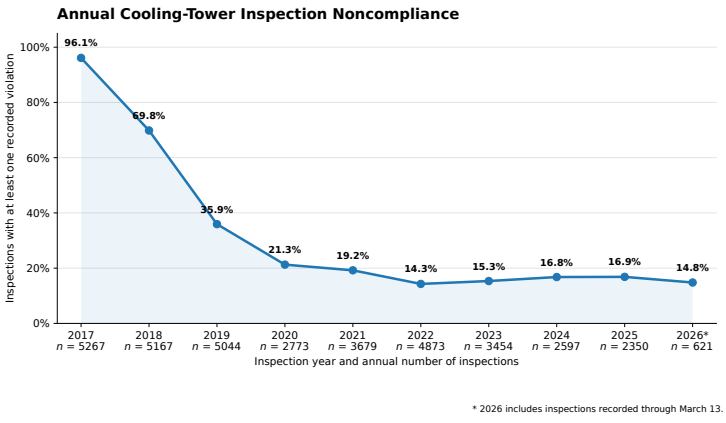


Figure 2. Annual number of cooling-tower inspections and percentage of inspections with at least one recorded violation.

3.4. Construction of Historical Predictors

We constructed historical predictors to represent the inspection, violation, and enforcement history available before each target inspection. For an inspection of system s conducted on date t, we calculated every historical predictor exclusively from inspections of the same system conducted on dates strictly earlier than t. We first ordered the inspection-

level records by System_ID and Inspection_date. Because the preceding preprocessing produced no duplicate System_ID–Inspection_date combinations, every preceding record represented a distinct earlier inspection episode.

We created four types of historical summaries. First, we shifted selected inspection-level fields by one position within each system to obtain information from the immediately preceding inspection. We also calculated the number of days between that inspection and the target inspection. Second, we calculated cumulative counts over all earlier inspections while excluding the target inspection. Third, we measured the elapsed time since the most recent earlier inspection with a violation and the number of consecutive noncompliant inspections immediately preceding the target inspection. Fourth, we calculated recent-history predictors over the interval from 365 days before the target date up to, but not including, the target date. Table 1 defines the 19 historical predictors created for each eligible target inspection.

Table 1. Historical predictors calculated from inspections conducted before each target inspection

Predictor	Definition
Previous_number_of_violations	Number of violations recorded at the immediately preceding inspection.
Previous_critical_violations	Number of Critical violations recorded at the immediately preceding inspection.
Previous_PHH_violations	Number of PHH violations recorded at the immediately preceding inspection.
Previous_inspection_type	Type of the immediately preceding inspection, classified as Cycle or Non-Cycle.
Days_since_previous_inspection	Number of calendar days between the immediately preceding inspection and the target inspection.
Total_previous_inspections	Number of inspections recorded for the system before the target inspection.
Total_previous_noncompliant_inspections	Number of earlier inspections at which inspectors recorded at least one violation.
Cumulative_previous_violations	Total number of violations recorded across all earlier inspections.
Cumulative_previous_critical_violations	Total number of Critical violations recorded across all earlier inspections.
Cumulative_previous_PHH_violations	Total number of PHH violations recorded across all earlier inspections.
Cumulative_previous_noncycle_inspections	Number of earlier inspections classified as Non-Cycle.
Days_since_last_violation	Number of calendar days since the most recent earlier inspection with at least one recorded violation; set to 3343 when no earlier recorded violation existed.
Consecutive_previous_noncompliant_inspections	Length of the uninterrupted sequence of noncompliant inspections immediately preceding the target inspection.
Previous_inspections_last_365_days	Number of earlier inspections conducted on or after the date 365 days before the target inspection date.
Previous_noncompliant_inspections_last_365_days	Number of earlier inspections with at least one recorded violation during the last 365 days before the target inspection date.
Previous_violations_last_365_days	Total number of violations recorded during the last 365 days before the target inspection date.
Previous_number_of_summonses	Number of distinct summons numbers recorded at the immediately preceding inspection.
Cumulative_previous_summonses	Sum of the inspection-level summons counts across all earlier inspections.
Total_previous_inspections_with_summons	Number of earlier inspections at which at least one summons was recorded.

After completing the original model analyses, we calculated Spearman correlations between all pairs of the 19 historical predictors in both development datasets. Using

$|\rho| \geq 0.80$ to summarize high correlations, we found 10 pairs in the grouped-random data and 9 in the chronological data. This analysis helped interpret permutation importance. All predictors were retained, and the models were not retuned. Full results are provided in Supplementary Figure S1 and the accompanying tables.

We retained both cumulative and recent-history predictors because they represented different temporal aspects of regulatory performance. For example, Cumulative_previous_violations measured the total recorded violation burden over the system's observed history, whereas Previous_violations_last_365_days measured its violation burden during the year immediately preceding the target inspection. Similarly, Total_previous_noncompliant_inspections measured the number of earlier inspections with at least one recorded violation, whereas Cumulative_previous_violations measured the total number of individual violations recorded across those inspections.

We treated the summons predictors as enforcement-history indicators. These predictors may reflect both the seriousness of earlier violations and the regulatory response to those violations. We therefore used them as predictive features but did not interpret their predictive relationships as causal effects of enforcement on subsequent compliance.

The first recorded inspection of each system had no earlier inspection history. We therefore excluded the 6162 first recorded inspections from the primary history-based modeling cohort. The resulting cohort contained 29,663 target inspections from 5668 systems, with each inspection having at least one earlier recorded inspection. All immediately preceding inspection fields were available for every observation in this cohort.

The field Days_since_last_violation remained structurally undefined for 1704 target inspections because the corresponding systems had earlier inspections but no previously recorded violation. The observation period spanned 3342 days. We therefore replaced these undefined values with 3343 days, one day greater than the longest possible elapsed time within the observation period. This sentinel represented the absence of any recorded violation in the system's observed inspection history rather than an accidental missing value. No other historical predictor contained a missing value in the modeling cohort.

4. Data Modeling

4.1. Common Experimental Protocol

We evaluated logistic regression, XGBoost, and TabM under two evaluation designs: system-grouped random evaluation and chronological evaluation. The combination of three algorithms and two evaluation designs produced six algorithm–evaluation configurations. Every configuration started from the same modeling cohort and used the same binary outcome, 19 historical predictors, and performance measures. The two evaluation designs partitioned the modeling cohort differently, as described in Sections 4.2 and 4.3.

We conducted the analyses using Python 3.11.5, pandas 2.0.0, NumPy 1.26.3, scikit-learn 1.6.0, XGBoost 3.2.0, PyTorch 2.1.1, TabM 0.0.3, and Matplotlib 3.9.3. We used 2026 as the master random seed.

4.1.1. Prediction Task and Model Inputs

For each target inspection, we predicted Any_violation, which equaled 1 when inspectors recorded at least one violation at that inspection and 0 otherwise. The modeling cohort contained 29,663 target inspections with at least one earlier recorded inspection. Inspectors recorded at least one violation at 8547 target inspections, corresponding to an outcome prevalence of 28.81%.

We supplied each algorithm with the 19 historical predictors defined in Table 1. We used System_ID for system-grouped partitioning and internal validation in the grouped-

random evaluation, and we used Inspection_date for chronological partitioning. We did not provide either field to the predictive algorithms.

The only categorical predictor was Previous_inspection_type, which we encoded as 0 for Cycle and 1 for Non-Cycle. The remaining 18 predictors were numerical counts or elapsed-time measures. As described in Section 3.4, the predictors contained no missing values.

4.1.2. Model Development and Test-Set Protection

For each evaluation design, we first designated a held-out test partition and used the remaining observations as the development dataset. We excluded the test observations from internal validation, model selection, estimation of data-dependent preprocessing quantities, and fitting of the final models. We used the test predictors only as inputs to the final models fitted using the development data.

We performed all model selection within the development dataset. For logistic regression and XGBoost, we fitted and compared models under alternative hyperparameter configurations. Hyperparameters are settings specified before fitting an individual model, such as regularization strength, tree depth, learning rate, and number of boosting trees. In contrast, model parameters—including logistic-regression coefficients, tree-split thresholds and leaf weights, and neural-network weights—were estimated from the corresponding training observations. We prespecified the TabM architecture and optimization settings rather than comparing alternative TabM configurations; internal validation selected only its number of training epochs.

For each internal training–validation split, we estimated all data-dependent preprocessing quantities and model parameters using only the training observations. We then applied the fitted preprocessing and model to the corresponding validation observations. We used validation average precision to select the logistic-regression and XGBoost hyperparameters and the TabM training duration according to the algorithm- and design-specific procedures described in the following subsections.

After completing model selection, we re-estimated the required preprocessing quantities from the complete development dataset and fitted a new final model using all development observations. For logistic regression and XGBoost, we used the selected hyperparameter configuration. For TabM, we used the prespecified configuration and the selected number of training epochs. We generated test predictions and calculated the final performance measures only after fixing and fitting the final model.

We did not use the held-out test outcomes to choose the predictors, preprocessing methods or quantities, logistic-regression or XGBoost hyperparameters, TabM configuration or training duration, or inspection- capacity levels. Separating model development from final test evaluation reduced the risk of optimistic selection bias that can arise when the same observations influence both model selection and performance estimation [26].

For each test inspection, we constructed the predictors using only information available before its inspection date. Therefore, the predictors for a later test inspection could include the known outcome of an earlier test inspection of the same system. We kept the fitted model unchanged throughout the test evaluation.

Within each evaluation design, all three algorithms used the same development and test partitions. They therefore generated predictions for the same test inspections, allowing direct descriptive comparisons on a common test sample.

4.1.3. Model-Input Preprocessing and Class-Distribution Handling

We estimated every data-dependent preprocessing quantity exclusively from the training observations used in the corresponding model fit. For each internal training–

validation split, we fitted the preprocessing procedure using only the training portion and applied the fitted transformation unchanged to the validation portion [27].

For logistic regression and TabM, we standardized the 18 nonbinary predictors and retained Previous_inspection_type on its original zero–one scale. For predictor j and inspection i, we calculated

$$z_{ij} = \frac{x_{ij} - \mu_{j,\text{train}}}{s_{j,\text{train}}}, \tag{1}$$

where x_{ij} is the original predictor value, $\mu_{j,\text{train}}$ is the mean calculated from the relevant training observations, and

$$s_{j,\text{train}} = \begin{cases} \sigma_{j,\text{train}}, & \text{if } \sigma_{j,\text{train}} > 0, \\ 1, & \text{if } \sigma_{j,\text{train}} = 0. \end{cases} \tag{2}$$

Here, $\sigma_{j,\text{train}}$ is the training-set standard deviation calculated with divisor n_{train}, equivalently `ddof = 0`, where `ddof` stands for Delta Degrees of Freedom. This procedure corresponds to `StandardScaler` in `scikit-learn` [27].

After completing internal model selection, we re-estimated the scaling quantities using the complete development dataset for the corresponding evaluation design. We then fitted the final logistic-regression and TabM models using the transformed development predictors and applied the development-derived transformation unchanged to the test predictors.

We supplied XGBoost with the encoded predictors on their unstandardized scales. Standardization was unnecessary because the decision-tree components of XGBoost partition observations through threshold comparisons and therefore depend on the ordering of predictor values rather than their measurement scales [28].

We fitted all three algorithms without class-dependent weighting or class-balancing resampling. We did not apply random oversampling, majority-class undersampling, or synthetic-data generation.

4.1.4. Performance Measures

Let N denote the number of test inspections, let $y_i \in \{0, 1\}$ denote the observed value of Any_violation, let $p_i \in [0, 1]$ denote the model's predicted probability of noncompliance, and let v_i denote the observed value of Number_of_violations for inspection i.

Precision, Recall, and Average Precision

At a prediction threshold t, we classified inspection i as noncompliant when $p_i \geq t$. We calculated precision as

$$\text{Precision}(t) = \frac{\text{TP}(t)}{\text{TP}(t) + \text{FP}(t)} \tag{3}$$

and recall as

$$\text{Recall}(t) = \frac{\text{TP}(t)}{\text{TP}(t) + \text{FN}(t)} \tag{4}$$

where $\text{TP}(t)$, $\text{FP}(t)$, and $\text{FN}(t)$ denote the numbers of true-positive, false-positive, and false-negative classifications at threshold t [29].

We used average precision as the model-selection criterion: it selected the logistic-regression and XGBoost hyperparameters and the TabM training duration. We also reported average precision as a threshold-independent measure of test-set ranking performance. Let Prec_m and Rec_m denote precision and recall at the mth distinct prediction threshold.

We ordered the thresholds so that recall was nondecreasing and defined $\mathrm{Rec}_0 = 0$. We calculated average precision as

$$\mathrm{AP} = \sum_{m=1}^{M} (\mathrm{Rec}_m - \mathrm{Rec}_{m-1})\,\mathrm{Prec}_m\,. \tag{5}$$

This is the definition implemented by `average_precision_score` in `scikit-learn`; it differs from calculating trapezoidal area under a precision–recall curve [27,30]. We used test-set noncompliance prevalence as the constant-score and no-skill reference for average precision. Because this reference depends on outcome prevalence, average-precision values from test datasets with different prevalences should not be compared without considering their corresponding reference levels [31].

Area Under the Receiver Operating Characteristic Curve

We quantified overall discrimination using the area under the receiver operating characteristic curve (AUROC). At each threshold τ_k, we classified inspection i as noncompliant when $p_i \geq \tau_k$ and calculated

$$\mathrm{TPR}_k = \frac{\mathrm{TP}_k}{\mathrm{TP}_k + \mathrm{FN}_k}, \quad \mathrm{FPR}_k = \frac{\mathrm{FP}_k}{\mathrm{FP}_k + \mathrm{TN}_k}. \tag{6}$$

The ROC curve plots FPR_k on the horizontal axis and TPR_k on the vertical axis across the complete sequence of thresholds. For the empirical ROC points, ordered by increasing false-positive rate and with $(\mathrm{FPR}_0, \mathrm{TPR}_0) = (0, 0)$, we calculated AUROC using trapezoidal integration:

$$\mathrm{AUROC} = \sum_{k=1}^{K} (\mathrm{FPR}_k - \mathrm{FPR}_{k-1}) \frac{\mathrm{TPR}_k + \mathrm{TPR}_{k-1}}{2}. \tag{7}$$

An AUROC of 0.5 represents chance-level discrimination, whereas an AUROC of 1 represents perfect discrimination [32].

Probabilistic Accuracy

We assessed overall probabilistic accuracy using the Brier score,

$$\mathrm{Brier} = \frac{1}{N} \sum_{i=1}^{N} (p_i - y_i)^2. \tag{8}$$

The Brier score ranges from 0 to 1 (given $y_i \in \{0, 1\}$), with smaller values indicating better probabilistic predictions [33].

We also calculated logarithmic loss as

$$\mathrm{LogLoss} = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)], \tag{9}$$

where log denotes the natural logarithm. Lower values of logarithmic loss indicate better probabilistic predictions.

For both scores, we compared each fitted model with a constant-probability reference that assigned the complete development-dataset noncompliance prevalence to every test inspection. We estimated this reference probability from the development outcomes rather than the held-out test outcomes.

Performance Under Constrained Inspection Capacity

To evaluate inspection prioritization, we ranked test inspections in descending order of p_i. We resolved exact probability ties by sorting first by ascending System_ID and then by ascending Inspection_date. This deterministic rule did not use the observed test outcome.

For each inspection-capacity level $c \in \{0.10, 0.20, 0.30\}$, we selected

$$n_c = \lceil cN \rceil \tag{10}$$

highest-ranked inspections. Let S_c denote the selected set, and define the total numbers of noncompliant inspections and recorded violations in the test dataset as

$$N_+ = \sum_{i=1}^{N} y_i, \qquad V = \sum_{i=1}^{N} v_i. \tag{11}$$

We calculated the numbers of noncompliant inspections and recorded violations identified within the selected set as

$$\mathrm{TP}_c = \sum_{i \in S_c} y_i, \qquad V_c = \sum_{i \in S_c} v_i. \tag{12}$$

We then calculated

$$\mathrm{Precision@}\, c = \frac{\mathrm{TP}_c}{n_c}, \qquad \mathrm{Recall@}\, c = \frac{\mathrm{TP}_c}{N_+}. \tag{13}$$

Precision at capacity represents the proportion of selected inspections that were noncompliant. Recall at capacity represents the proportion of all noncompliant test inspections contained in the selected set.

We calculated test-set noncompliance prevalence as

$$\pi = \frac{N_+}{N} \tag{14}$$

and defined lift as

$$\mathrm{Lift@}\, c = \frac{\mathrm{Precision@}\, c}{\pi}. \tag{15}$$

A lift of 1 equals the expected precision under uniform random selection, whereas a lift greater than 1 indicates that the ranking concentrated noncompliant inspections above that reference.

We calculated violation coverage as

$$\mathrm{ViolationCoverage@}\, c = \frac{V_c}{V}. \tag{16}$$

Thus, V_c represents the number of recorded violations identified, whereas violation coverage represents the proportion of all test-set violations contained in the selected inspections. Under uniform random selection of n_c inspections, expected recall and expected violation coverage both equal n_c/N, expected precision equals π, and expected lift equals 1.

We also calculated cumulative gain across the complete risk ranking. Let $y_{(r)}$ denote the outcome of the inspection at rank r. At each $k \in \{1, \ldots, N\}$, cumulative gain is

$$G\left(\frac{k}{N}\right) = \frac{\sum_{r=1}^{k} y_{(r)}}{N_+} \tag{17}$$

The random-ranking reference is the diagonal $G(x) = x$, while the perfect-ranking reference is $G(x) = \min(x/\pi, 1)$.

4.1.5. Simple History-Rule Comparators

To determine whether the fitted models added value beyond transparent use of inspection history, we evaluated four rules that required no model fitting. We ranked inspections by (i) whether the immediately preceding inspection was noncompliant, (ii) the number of violations at the immediately preceding inspection, (iii) the cumulative number of violations before the target inspection, and (iv) the number of days since the most recent previous violation. The first three rules assigned higher priority to larger values; the fourth assigned higher priority to smaller values.

We applied each rule separately to the grouped-random and chronological test datasets and evaluated the same capacity levels, precision, recall, lift, number of violations identified, and violation coverage used for the fitted models. These rules are analytical comparators; we do not assume that they represent an existing NYC inspection-scheduling policy.

The history rules frequently assigned the same priority score to multiple inspections. A tie sometimes crossed the capacity cutoff, meaning that the remaining capacity was insufficient to select every inspection with the boundary score. In this situation, we included all inspections assigned strictly higher priority than the boundary score and evaluated the remaining places as a uniform random selection from the tied boundary group.

Let A_c denote the inspections with priority strictly above the boundary, and let T_c denote the inspections tied at the boundary. After selecting all inspections in A_c, the number of places remaining was $n_c - |A_c|$. Thus, each inspection in T_c had the same selection probability

$$r_c = \frac{n_c - |A_c|}{|T_c|}. \tag{18}$$

Thus, we calculated the expected numbers of noncompliant inspections and violations identified at capacity c as

$$\mathbb{E}(\mathrm{TP}_c) = \sum_{i \in A_c} y_i + r_c \sum_{i \in T_c} y_i, \tag{19}$$

and

$$\mathbb{E}(V_c) = \sum_{i \in A_c} v_i + r_c \sum_{i \in T_c} v_i. \tag{20}$$

4.1.6. Permutation-Based Feature Importance

After completing model fitting and performance evaluation, we used permutation-based feature importance to quantify how strongly each fitted model's test-set average precision depended on each predictor [34]. We first calculated average precision on the unmodified test dataset. For each predictor j, we then randomly permuted its test-set values while retaining all other predictors unchanged, generated new predictions using the already fitted model, and recalculated average precision. We did not refit the model.

We repeated the permutation 50 times for each predictor and defined importance as

$$\mathrm{PFI}_j = \mathrm{AP}_{\mathrm{original}} - \frac{1}{50} \sum_{r=1}^{50} \mathrm{AP}_{j,r}^{\mathrm{permuted}}. \tag{21}$$

We applied the same permutation-importance procedure to all three models. For logistic regression and XGBoost, we used the `permutation_importance` function provided by `scikit-learn`. Because our TabM implementation was not directly compatible with this function, we performed the same calculation explicitly for the TabM model [27].

A larger positive value indicates greater dependence of the fitted model's test-set ranking on that predictor. A value near zero indicates that permutation produced little detectable mean change in average precision, whereas a negative value indicates that mean average precision was higher after permutation. The results are specific to the fitted model,

test dataset, and scoring measure. The permutation results did not influence preprocessing or model selection. With correlated predictors, another predictor may supply similar information, attenuating the measured importance of the shuffled variable.

4.1.7. Computational Environment and Timing

We measured computational cost in subsequent timing reruns of the original procedures, with one rerun per model and evaluation design. We retained the original data partitions, model settings, and random seed. All six runs used an Intel Core i5-1335U CPU with 7.53 GiB of physical RAM and Windows (build 26200). The reruns used the software versions reported at the start of the common experimental protocol, together with SciPy 1.10.1. The six experiments were run sequentially. Table 2 reports elapsed running times for model selection and final training separately. Times include preprocessing, validation, and search overhead within the measured stages.

Table 2. Elapsed model selection and final training times (in seconds) from CPU timing reruns.

Design	Model	Selection Fits	Selection (s)	Final Training (s)	Total (s)
Grouped random	Logistic regression	102	15.5	0.3	15.8
Grouped random	XGBoost	288	238.6	7.1	245.7
Grouped random	TabM	1	382.4	377.3	759.7
Chronological	Logistic regression	85	5.9	0.3	6.2
Chronological	XGBoost	240	100.6	1.4	102.0
Chronological	TabM	5	1267.7	102.6	1370.3

4.1.8. Sensitivity to Same-Day Record Handling

We tested whether removing same-day Non-Cycle records affected model performance. For each system and date with both inspection types, we combined their distinct violations and summonses into one episode and kept its type as Cycle. We rebuilt the historical predictors and retrained all three models under both cleaning rules, using the original data splits, selected model settings, and random seed. We compared average precision, AUROC, Brier score, log loss, and performance at 10%, 20%, and 30% inspection capacity. The results are summarized in Supplementary Table S1.

4.2. Grouped Random Evaluation

The grouped-random evaluation assessed whether models trained on one set of cooling-tower systems could generalize to repeat inspections from different systems. For this evaluation design, we assigned every target inspection from the same System_ID to one fold. Consequently, no system appeared in both the development and test datasets. This separation prevented repeated target inspections from the same system from appearing on opposite sides of the development–test split and followed recommended validation practice for hierarchically structured observations [9]. The test systems were not represented during model development, although each test inspection retained the historical predictors constructed from information available before its inspection date. This design assessed generalization across systems rather than across time; Section 4.3 addresses temporal generalization.

4.2.1. Partition Construction and Internal Validation

We first sorted the modeling cohort by System_ID and Inspection_date. We then created one seven-fold assignment using `StratifiedGroupKFold` with seven folds, shuffling enabled, and random seed 2026. We supplied System_ID as the grouping variable and Any_violation as the stratification target. Thus, the procedure kept all target inspections

from one system in the same fold while attempting to maintain approximately the same noncompliance prevalence across the seven folds. Because all inspections from each system had to remain together in one fold, the folds could not be made exactly equal in size or noncompliance prevalence [27]. We used Any_violation only to construct the stratified folds; we did not use it as a model predictor.

Before fitting or tuning any model, we designated fold 7 as the held-out test partition. We excluded this fold from the estimation of preprocessing quantities, hyperparameter or training-duration selection, and model fitting. Folds 1–6 constituted the development dataset.

For logistic regression and XGBoost, we used the six development folds in six-fold grouped cross-validation. For each candidate hyperparameter configuration, we fitted the model six times. In each fit, we used five folds for training and the remaining fold for validation, rotating the validation fold until every development fold had served once in that role. Therefore, each development system contributed to five training fits and one validation fit. Within each fit, we estimated all model parameters and, where required, all data-dependent preprocessing quantities exclusively from the five training folds. We then applied the fitted preprocessing and model to the held-out validation fold. We calculated average precision separately for the six validation folds and selected the configuration with the highest unweighted arithmetic mean of the six values.

We used a different internal-validation procedure for TabM because we prespecified its architecture and optimization settings rather than comparing alternative configurations. We trained a TabM validation model on development folds 1–5, kept development fold 6 separate from model fitting, and used its average precision to select the number of training epochs, as described in Section 4.2.4.

After completing hyperparameter selection for logistic regression and XGBoost and training-duration selection for TabM, we re-estimated any required preprocessing quantities and fitted each final model using all 25,446 development inspections. We used the selected hyperparameter configuration for logistic regression and XGBoost and the prespecified configuration and selected number of epochs for TabM. We then evaluated each final model once on the same 4217 held-out test inspections.

4.2.2. Logistic Regression

Model Specification

We used ℓ_2-regularized binary logistic regression as the linear probabilistic model for predicting Any_violation. For target inspection i, let $y_i \in \{0, 1\}$ denote the observed value of Any_violation, where $y_i = 1$ indicates that the inspection identified at least one violation. Let $\mathbf{x}_i$ denote the 19-dimensional model-input vector obtained after applying the preprocessing procedure described in Section 4.1.3. Logistic regression models the conditional probability of a binary outcome using the logistic function [35]. We defined the linear predictor as

$$\eta_i = \beta_0 + \mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta}, \tag{22}$$

where β_0 is the intercept, and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{19})^{\mathsf{T}}$ is the vector of predictor coefficients. The estimated probability of noncompliance was

$$p_i = \Pr(y_i = 1 \mid \mathbf{x}_i) = \frac{1}{1 + \exp(-\eta_i)}. \tag{23}$$

We fitted the model with an ℓ_2 penalty, which shrinks coefficient estimates toward zero and can reduce coefficient instability while retaining all predictors in the model [36]. Because every training inspection received equal weight, the model minimized

$$\mathcal{L}(\beta_0, \boldsymbol{\beta}; C) = -\frac{1}{n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] + \frac{1}{2n_{\text{tr}}C} \sum_{j=1}^{19} \beta_j^2, \tag{24}$$

where n_{tr} is the number of training inspections. The first term is the mean binary logarithmic loss, and the second term is the ℓ_2-penalty contribution. The intercept β_0 was not penalized. The positive hyperparameter C is the inverse regularization strength: smaller values impose stronger coefficient shrinkage, whereas larger values impose weaker shrinkage. This objective corresponds to the regularized binary logistic-regression objective implemented in `scikit-learn` [27].

Implementation and Preprocessing

We implemented the model using `LogisticRegression` with `penalty=l2`, `solver=lbfgs`, `fit_intercept=True`, and `class_weight=None`. We supplied no observation-level sample weights. We set the maximum number of optimization iterations to 20,000 and the convergence tolerance to 10^{-7}. The `lbfgs` solver implements the limited-memory Broyden–Fletcher–Goldfarb–Shanno optimization algorithm [37].

We combined the `ColumnTransformer` and logistic-regression classifier within a single `Pipeline`. In each cross-validation fit, the pipeline estimated the standardization quantities using only the five training folds and applied the fitted transformation unchanged to the held-out validation fold. This implementation kept the preprocessing procedure within each cross-validation fit [27].

Regularization Selection

We tuned only the inverse regularization parameter C. We examined 17 logarithmically spaced candidate values defined by

$$C_m = 10^{-4+0.5m}, \qquad m \in \{0, 1, \ldots, 16\}, \tag{25}$$

which ranged from 10^{-4} to 10^4 and covered settings from strong to weak regularization.

We evaluated each candidate using the six fixed grouped cross-validation folds described in Section 4.2.1. For each candidate value, we fitted the complete preprocessing-and-model pipeline six times, using five development folds for training and the remaining fold for validation. The search therefore required $17 \times 6 = 102$ cross-validation fits.

We calculated average precision separately for each validation fold and selected the value of C with the highest unweighted arithmetic mean across the six folds. We used only the development dataset for this selection and kept the independent test dataset separate from hyperparameter selection [26]. We then followed the final refitting and held-out test-evaluation procedure described in Section 4.2.1.

4.2.3. XGBoost

Model Specification

XGBoost constructs an ensemble of decision trees sequentially. Each new tree uses the errors remaining from the existing ensemble to improve the model. For inspection i, we represented the ensemble score and predicted probability of noncompliance as

$$\begin{aligned} F_K(\mathbf{x}_i) &= F_0 + \eta \sum_{k=1}^{K} f_k(\mathbf{x}_i), \\ p_i &= \frac{1}{1 + \exp[-F_K(\mathbf{x}_i)]}, \end{aligned} \tag{26}$$

where $\mathbf{x}_i$ contains the predictors for inspection i, F_0 is the initial ensemble score, K is the number of boosting rounds, η is the learning rate, and $f_k(\mathbf{x}_i)$ is the contribution of tree k. The learning rate scales the contribution of every new tree before XGBoost adds it to the ensemble. The logistic transformation converts the final ensemble score into a predicted probability between zero and one.

XGBoost selects the structure and leaf scores of each new tree using the first- and second-order derivatives of the loss function. This sequential tree-building process allows the model to represent nonlinear predictor effects and interactions without requiring us to specify their functional forms in advance [38,39].

For the binary outcome, XGBoost minimized the negative log-likelihood together with penalties controlling tree complexity and leaf scores. We express the regularized objective as

$$\begin{aligned} \mathcal{J} = &\sum_{i=1}^{n_{\text{tr}}} [-y_i \log(p_i) - (1 - y_i) \log(1 - p_i)] \\ &+ \sum_{k=1}^{K} \left[\gamma T_k + \frac{\lambda}{2} \sum_{j=1}^{T_k} w_{kj}^2 + \alpha \sum_{j=1}^{T_k} |w_{kj}| \right], \end{aligned} \tag{27}$$

where n_{tr} is the number of training inspections in the corresponding model fit, y_i equals one for a noncompliant inspection and zero otherwise, T_k is the number of terminal leaves in tree k, and w_{kj} is the score assigned to leaf j in tree k. The parameter γ penalizes the number of terminal leaves, while α and λ control the ℓ_1 and ℓ_2 regularization of the leaf scores, respectively [38,39].

Implementation and Preprocessing

We implemented XGBoost 3.2.0 using `XGBClassifier` and used scikit-learn 1.6.0 to conduct the randomized search and grouped cross-validation. We set `booster=gbtree`, `objective=binary:logistic`, `tree_method=hist`, and `random_state=2026`. The histogram-based tree-building method grouped continuous predictor values into candidate split bins to construct the trees efficiently.

Consistent with the common preprocessing procedure described in Section 4.1.3, we supplied the 19 encoded predictors on their original, unstandardized scales. Tree-based models do not require predictors to share a common measurement scale because their splits depend on the ordering of predictor values rather than distances between values.

We set `scale_pos_weight=1` and supplied no observation-level sample weights. Therefore, we did not assign additional weight to the positive class, and every training inspection had the same base weight. We also applied no outcome resampling or synthetic-data generation. We left `base_score` at its XGBoost default, under which the software estimated the initial model intercept from the outcome values in the training observations used in each fit [39].

Hyperparameter Selection

We selected nine XGBoost hyperparameters jointly using randomized search. One hyperparameter configuration consisted of one sampled value for each of the nine hyperparameters listed in Table 3. Thus, the search evaluated complete combinations of the nine settings rather than selecting each setting separately. We specified the values and ranges in Table 3 before examining the independent test results.

We used `RandomizedSearchCV` to sample 48 complete configurations. We set the random-search seed to 2026 so that the same configurations could be reproduced. Randomized search allowed us to examine combinations drawn from a large search space without evaluating every possible combination [27,40].

This was a fixed search budget. The selected configuration was the best among those tested, based on mean validation average precision. Better settings may exist elsewhere in the search space.

Table 3. Values and ranges used to sample the 48 XGBoost hyperparameter configurations under the grouped-random evaluation.

Hyperparameter	Function	Sampling Procedure
`n_estimators`	Number of boosting rounds	Integer sampled uniformly from 200 through 1200
`learning_rate`	Step-size shrinkage applied to each new tree's leaf scores	Value sampled log-uniformly from 0.01 to 0.20
`max_depth`	Maximum depth of each tree	Integer sampled uniformly from 2 through 7
`min_child_weight`	Minimum sum of second-order instance weights required in a child node	Value sampled log-uniformly from 0.50 to 20.00
`gamma`	Minimum reduction in the objective required to split a node	One value sampled uniformly from $\{0, 0.05, 0.10, 0.25, 0.50, 1, 2, 5\}$
`subsample`	Fraction of training inspections sampled in each boosting round	Value sampled uniformly from 0.65 to 1.00
`colsample_bytree`	Fraction of predictors sampled when constructing each tree	Value sampled uniformly from 0.65 to 1.00
`reg_alpha`	Strength of the ℓ_1 penalty on leaf scores	Value sampled log-uniformly from 10^{-5} to 10
`reg_lambda`	Strength of the ℓ_2 penalty on leaf scores	Value sampled log-uniformly from 0.10 to 100

Uniform sampling assigned equal probabilities to intervals of equal width. For the two integer-valued hyperparameters, it assigned equal probability to each integer in the specified range. Log-uniform sampling operated on the logarithmic scale, giving equal sampling opportunity to multiplicative intervals and allowing the search to examine values across several orders of magnitude. For `gamma`, the search selected each value in the specified list with equal probability.

We passed the six fixed grouped cross-validation splits described in Section 4.2.1 to `RandomizedSearchCV`. We evaluated every one of the 48 configurations using all six splits. For each configuration, we fitted XGBoost six times. In each fit, we trained the model using five development folds and evaluated it on the remaining validation fold. Therefore, the search required a total of 288 cross-validation fits.

For every fit, we calculated average precision on the corresponding validation fold. Each configuration therefore produced six validation average-precision values. We calcu-

lated the unweighted arithmetic mean of validation average precision across the six folds. We selected the configuration with the highest mean validation average precision.

4.2.4. TabM

Model Specification

TabM is a neural-network architecture for tabular data that combines a multilayer perceptron with parameter-efficient ensembling. A conventional deep ensemble fits several independent networks and averages their predictions. TabM instead represents K ensemble members within one network. The members share most of the network parameters, while small member-specific parameters allow them to produce different representations and predictions. TabM trains these members jointly and processes them in parallel [41,42].

For inspection i, let p_{im} denote the probability of noncompliance produced by ensemble member m. We calculated the final predicted probability by averaging the K member probabilities:

$$p_i = \frac{1}{K} \sum_{m=1}^{K} p_{im}. \tag{28}$$

For binary classification, this calculation averages the probabilities rather than the members' untransformed outputs, as recommended by the official TabM implementation [42].

Implementation and Prespecified Configuration

We implemented TabM with version 0.0.3 of the official `tabm` package. We used the 19 preprocessed historical predictors as inputs to TabM. We prespecified the architecture and optimization settings shown in Table 4. To limit CPU computation, we evaluated one prespecified TabM configuration and used validation data to select the number of training epochs. We did not search alternative architectures or optimization settings. The measured cost of this procedure is reported in Section 4.1.7. The results therefore describe this configuration and do not establish that it is optimal for the dataset.

Table 4. Prespecified TabM configuration under the grouped-random evaluation design.

Setting	Value	Purpose
Ensemble members	32	Number of member-specific probabilities produced for each inspection and averaged to obtain the final probability.
Hidden blocks	2	Number of successive hidden processing blocks.
Units per hidden block	128	Width of the representation produced by each hidden block.
Dropout rate	0.10	Proportion of hidden-unit outputs randomly set to zero during training.
Optimizer	AdamW	Algorithm used to update the network parameters.
Learning rate	10^{-3}	Base step size used by AdamW for parameter updates.
Weight decay	10^{-4}	Strength of the AdamW regularization that reduced the magnitudes of the network parameters
Minibatch size	1024	Number of training inspections used for each parameter update.
Maximum epochs	75	Maximum number of complete passes through the training subset during the validation fit.
Early-stopping patience	10 epochs	Number of consecutive epochs allowed without a qualifying improvement in validation average precision.

Table 4. *Cont.*

Setting	Value	Purpose
Minimum AP improvement	10^{-5}	Increase in validation average precision required to count as an improvement.
Gradient-norm limit	1.0	Maximum permitted combined magnitude of the parameter gradients. If this magnitude exceeded the limit, all gradients were reduced proportionally before each AdamW update.
Random seeds	2026	Seed used by the random procedures during both the validation and final fits.
Class and observation weights	None	No additional weight was assigned to noncompliant inspections or to any individual inspection.

Preprocessing and Training-Duration Selection

We used the same grouped-random development and independent test datasets used for logistic regression and XGBoost. Within the development dataset, we used folds 1–5 as the training subset and fold 6 as the validation subset. We used fold 6 only to select the number of training epochs; we did not use it to compare alternative TabM configurations.

Before the validation fit, we fitted `StandardScaler` from scikit-learn 1.6 to the 18 nonbinary predictors using only folds 1–5. We used the fitted means and standard deviations to transform these predictors in both the training and validation subsets. We retained Previous_inspection_type in its binary 0/1 form without standardization [27].

During each minibatch update, the model produced one prediction for each inspection and ensemble member. For a minibatch $\mathcal{B}$, we calculated the binary logarithmic loss as

$$\mathcal{L}_{\mathcal{B}} = -\frac{1}{|\mathcal{B}|K} \sum_{i \in \mathcal{B}} \sum_{m=1}^{K} [y_i \log(p_{im}) + (1 - y_i) \log(1 - p_{im})], \quad (29)$$

where $y_i = 1$ denotes a noncompliant inspection, and $y_i = 0$ denotes a compliant inspection.

We trained the validation model for at most 75 epochs using seed 2026. After each epoch, we calculated the 32 member probabilities for every validation inspection, averaged them according to Equation (28), and calculated validation average precision. We recorded an improvement only when average precision exceeded the previously retained value by more than 10^{-5}. We recorded the epoch and model parameters whenever this condition was satisfied. We stopped training after 10 consecutive epochs without a qualifying improvement or after 75 epochs, whichever occurred first. We selected the recorded epoch as the final training duration.

4.3. Chronological Evaluation

The chronological evaluation assessed how models developed from earlier inspection records performed on later inspections, unlike the grouped-random evaluation, which assessed generalization to cooling-tower systems excluded from model development [9,43].

4.3.1. Split Construction

We used January 1, 2024, as the fixed boundary between model development and testing. We assigned inspections conducted before this date to the development dataset and inspections conducted on or after this date to the temporally held-out test dataset. This boundary retained the complete calendar years 2017–2023 for model development and reserved all inspections from 2024 onward for testing. We fixed the boundary before model fitting and performance assessment and did not select or alter it according to any algorithm's results. We applied neither random allocation nor outcome stratification.

Table 5 summarizes the resulting partitions. Noncompliance prevalence was 15.17 percentage points lower in the test dataset than in the development dataset.

Table 5. Chronological development and held-out test partitions.

Partition	Period	Inspections	Distinct Systems	Noncompliant Inspections	Prevalence	Recorded Violations
Development	21 February 2017– 29 December 2023	24,242	5555	7657	31.59%	32,845
Held-out test	2 January 2024 – 13 March 2026	5421	3981	890	16.42%	2753

We allowed cooling-tower systems to contribute inspections to both periods. In total, 3868 systems appeared in both datasets, representing 97.16% of the test systems. This overlap was intentional because the chronological design assessed prediction for later inspections rather than generalization exclusively to systems absent from model development.

4.3.2. Expanding-Window Temporal Validation and Model Selection

Within the chronological development dataset, we performed five iterations of expanding-window temporal validation for internal model selection. In every iteration, the training window contained only inspections conducted before the subsequent validation year, and the training window expanded as additional inspection records became available [9,44].

We used each calendar year from 2019 through 2023 as a validation window. In the first iteration, we trained on the eligible inspections from 2017–2018 and validated on the inspections from 2019. For each subsequent iteration, we added the preceding validation year to the training window and validated on the next calendar year. The final iteration therefore trained on inspections from 2017–2022 and validated on inspections from 2023.

We fixed 2019 as the first validation year before fitting any model. Had we started validation in 2018, the initial training window would have contained only 111 eligible inspections from 2017: 106 noncompliant inspections (95.50%) and 5 compliant inspections. This initial window would therefore have been both very small and highly imbalanced. Beginning validation in 2019 increased the initial training window to 4937 inspections.

Table 6 summarizes the five iterations. Because we defined the validation windows by calendar year rather than by an equal number of observations, the numbers of inspections and the noncompliance prevalences varied across validation years. This choice retained five one-year validation periods and kept inspections from the same calendar year together.

Table 6. Expanding-window temporal-validation iterations within the chronological development dataset.

Validation Iteration	Training Period	Training Inspections	Training Noncompliant Inspections (%)	Validation Year	Validation Inspections	Validation Noncompliant Inspections (%)
1	2017–2018	4937	3480 (70.49%)	2019	4876	1735 (35.58%)
2	2017–2019	9813	5215 (53.14%)	2020	2692	569 (21.14%)
3	2017–2020	12,505	5784 (46.25%)	2021	3555	683 (19.21%)
4	2017–2021	16,060	6467 (40.27%)	2022	4805	683 (14.21%)
5	2017–2022	20,865	7150 (34.27%)	2023	3377	507 (15.01%)

For logistic regression and XGBoost, we fitted every candidate hyperparameter configuration in each of the five iterations. Within each iteration, we estimated the applicable preprocessing quantities and model parameters using only the training window and then calculated average precision on the immediately following validation year. We used the unweighted arithmetic mean of the five annual average-precision values as the selection score.

Thus, each validation year contributed equally, irrespective of its number of inspections. We selected the configuration with the largest mean using the unrounded annual values.

For TabM, we did not compare alternative hyperparameter configurations. We used the same five iterations to select a common training duration for the prespecified configuration, as described in Section 4.3.5. After completing the algorithm-specific selection procedure, we followed the final-refitting and held-out test procedure.

4.3.3. Chronological Logistic-Regression Experiment

We applied the logistic-regression specification, preprocessing pipeline, and 17-value grid for the inverse regularization parameter C described in Section 4.2.2. The chronological experiment differed only in how we partitioned the data and performed internal validation. For each candidate value of C, we fitted the complete pipeline separately in each of the five expanding-window temporal-validation iterations described in Section 4.3.2. The search therefore required $17 \times 5 = 85$ training–validation fits. In each validation year, we calculated average precision. We selected C using the mean validation average precision. After selecting C, we followed the final-refitting and held-out test procedure.

4.3.4. Chronological XGBoost Experiment

We applied the XGBoost specification and hyperparameter sampling distributions described in Section 4.2.3 to the chronological development and test partitions described in Section 4.3.1. Using the same sampling distributions and random-search seed, we evaluated the same 48 candidate configurations used in the grouped-random experiment. We fitted each configuration separately in the five expanding-window temporal-validation iterations described in Section 4.3.2. The search therefore required $48 \times 5 = 240$ training–validation fits. In each validation year, we calculated average precision. We selected the configuration with the highest mean validation average precision. After selecting the hyperparameters, we followed the final-refitting and held-out test procedure.

4.3.5. Chronological TabM Experiment

We applied the prespecified TabM configuration described in Section 4.2.4 to the five expanding-window temporal-validation iterations described in Section 4.3.2. We used these iterations to select a common training duration for the prespecified configuration.

Within each iteration, we applied the training-window preprocessing procedure defined in Section 4.1.3. We initialized a new TabM model and trained it for all 75 candidate epochs. After every epoch, we calculated average precision on the subsequent validation year. For each epoch, we then calculated the unweighted arithmetic mean of the five annual average-precision values.

After completing the five fits, we scanned the mean validation average-precision values in epoch order. We retained an epoch only when its mean average precision exceeded the value at the previously retained epoch by more than 10^{-5}. We stopped the scan after 10 consecutive epochs without such an improvement and selected the most recently retained epoch. Thus, the rule selected one training duration from the five-year mean trajectory; it did not stop any of the five individual validation fits before epoch 75.

After selecting the training duration, we discarded the five validation models, re-estimated the preprocessing quantities using the complete chronological development dataset, and initialized a new TabM. We fitted this model to the complete development dataset for exactly the selected number of epochs. We then followed the held-out test procedure and calculated the performance measures.

To reduce the risk of overfitting, we selected the training duration using temporal validation and used dropout of 0.10 and AdamW weight decay of 10^{-4}. The held-out test data were used only for final evaluation.

5. Results

5.1. Grouped Random Evaluation Results

5.1.1. Logistic-Regression Results

Six-fold grouped cross-validation selected $C = 0.3162$, at which the mean validation average precision was 0.6378. We refitted the selected model using all 25,446 development inspections and evaluated it once on the 4217 inspections in the independent grouped-random test dataset. The test dataset contained 1295 noncompliant inspections, corresponding to a prevalence of 30.71%. Table 7 reports the four prespecified test-performance measures. The average-precision reference equals the test-set noncompliance prevalence, and the AUROC reference represents chance-level discrimination. For the Brier score and logarithmic loss, the reference assigned the development-set prevalence of 0.2850 to every test inspection.

Table 7. Performance of the logistic-regression model on the grouped-random test dataset.

Measure	Logistic Regression	Reference Value	Preferred Direction
Average precision	0.6695	0.3071	Higher
AUROC	0.7864	0.5000	Higher
Brier score	0.1594	0.2133	Lower
Logarithmic loss	0.4920	0.6179	Lower

The model's test average precision of 0.6695 exceeded the prevalence reference of 0.3071, and its AUROC of 0.7864 exceeded the chance reference of 0.5000. Figure 3 shows the corresponding receiver operating characteristic and precision–recall curves across the complete range of thresholds. Between its endpoints, the receiver operating characteristic curve remained above the chance diagonal, while the precision–recall curve remained above the prevalence reference over most of the recall range. The model also produced lower Brier score and logarithmic loss than the constant development-prevalence reference.

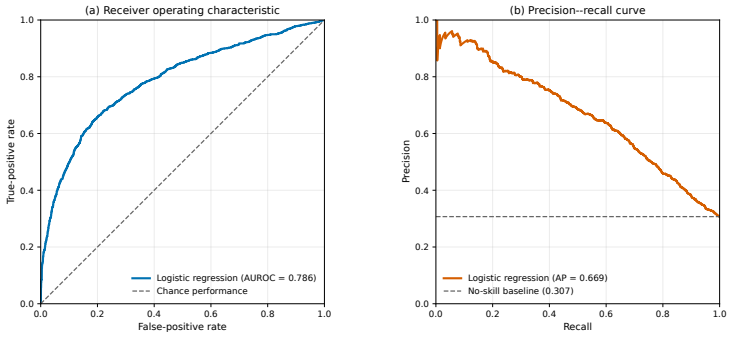


Figure 3. Discrimination of the logistic-regression model on the grouped-random test dataset.

The ranking concentrated noncompliant inspections and recorded violations within the highest-risk portions of the test dataset (Table 8). At the nominal 10% capacity level, the ranking selected 422 inspections, of which 344 were noncompliant. This yielded 81.52% precision, 26.56% recall, and a lift of 2.65. The selected inspections also contained 2267 of the 5356 recorded violations, corresponding to 42.33% violation coverage. As capacity increased from 10% to 30%, recall increased from 26.56% to 61.47%, and violation coverage increased from 42.33% to 74.03%, whereas precision decreased from 81.52% to 62.88%, and lift decreased from 2.65 to 2.05. Figure 4 extends the capacity analysis across the complete ranking. Between its endpoints, the cumulative-gain curve remained above the random-order reference.

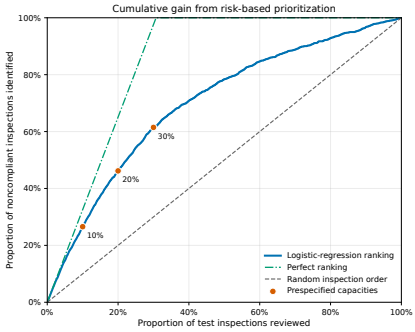


Figure 4. Cumulative gain from the logistic-regression risk ranking on the grouped-random test dataset.

Table 8. Capacity-constrained performance of the logistic regression on the independent grouped-random test dataset.

Capacity	Inspections Selected	Noncompliant Inspections Captured	Precision	Recall	Lift	Violations Captured	Violation Coverage
10%	422	344	81.52%	26.56%	2.65	2267	42.33%
20%	844	598	70.85%	46.18%	2.31	3309	61.78%
30%	1266	796	62.88%	61.47%	2.05	3965	74.03%

The test-set ranking was most sensitive to permutation of Previous_number_of_violations and Total_previous_inspections (Figure 5). Permuting these predictors reduced average precision by 0.2655 and 0.1354 on average, respectively.

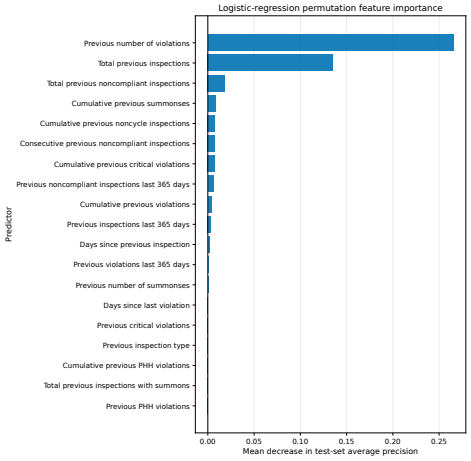


Figure 5. Permutation-based feature importance for the logistic-regression model on the grouped-random test dataset.

5.1.2. XGBoost Results

Six-fold grouped cross-validation selected an XGBoost configuration with the highest mean validation average precision 0.6587. We refitted the selected configuration using all 25,446 development inspections and evaluated it once on the 4217 inspections in the independent grouped-random test dataset. Table 9 reports the four prespecified test-performance measures and their reference values.

Table 9. Performance of the XGBoost model on the grouped-random test dataset.

Measure	XGBoost	Reference Value	Preferred Direction
Average precision	0.6832	0.3071	Higher
AUROC	0.8003	0.5000	Higher
Brier score	0.1539	0.2133	Lower
Logarithmic loss	0.4780	0.6179	Lower

The model's test average precision of 0.6832 exceeded the prevalence reference of 0.3071, and its AUROC of 0.8003 exceeded the chance reference of 0.5000. Its Brier score and logarithmic loss were also lower than those of the constant reference that assigned the development-set noncompliance prevalence of 0.2850 to every test inspection. Figure 6 shows the corresponding receiver operating characteristic and precision–recall curves. Together, these results indicate better risk ranking and lower probabilistic loss than the respective references.

The risk ranking concentrated noncompliant inspections and recorded violations within the highest-risk portions of the test dataset (Table 10). At the nominal 10% capacity level, the ranking selected 422 inspections, of which 353 were noncompliant. This yielded 83.65% precision, 27.26% recall, and a lift of 2.72. The selected inspections contained 2274 of the 5356 recorded violations, corresponding to 42.46% violation coverage. As capacity increased from 10% to 30%, recall increased from 27.26% to 62.93%, and violation coverage increased from 42.46% to 74.05%, whereas precision decreased from 83.65% to 64.38%, and lift decreased from 2.72 to 2.10.

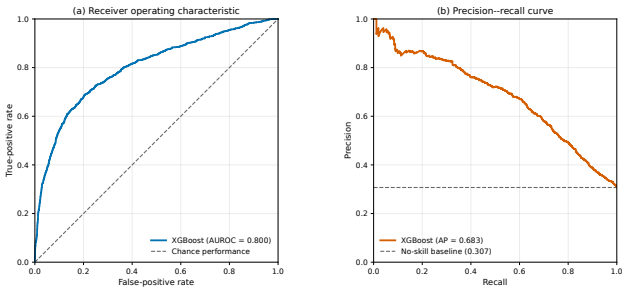


Figure 6. Discrimination of the XGBoost model on the grouped-random test dataset.

Table 10. Capacity-constrained performance of the XGBoost risk ranking on the grouped-random test dataset.

Capacity	Inspections Selected	Noncompliant Inspections Captured	Precision	Recall	Lift	Violations Captured	Violation Coverage
10%	422	353	83.65%	27.26%	2.72	2274	42.46%
20%	844	614	72.75%	47.41%	2.37	3327	62.12%
30%	1266	815	64.38%	62.93%	2.10	3966	74.05%

Figure 7 extends the capacity analysis across the complete test ranking. Between its endpoints, the cumulative-gain curve remained above the random-order reference.

The model's test-set average precision was most sensitive to permutation of Previous_number_of_violations and Total_previous_inspections (Figure 8). Permuting these predictors reduced average precision by 0.1263 and 0.0641 on average, respectively.

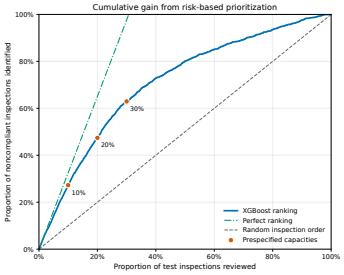


Figure 7. Cumulative gain from the XGBoost risk ranking on the grouped-random test dataset.

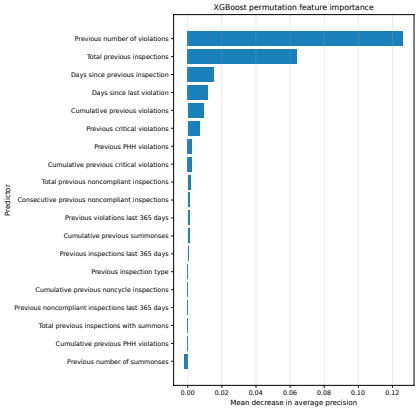


Figure 8. Permutation-based feature importance for the XGBoost model on the grouped-random test dataset.

5.1.3. TabM Results

TabM achieved its highest validation average precision of 0.6686 at epoch 60. We therefore fitted the final model to the complete development dataset for 60 epochs and evaluated it on the independent test dataset.

On the 4217 test inspections, TabM achieved an average precision of 0.6852 and an AUROC of 0.7982. Its Brier score was 0.1549, compared with 0.2133 for the uninformative reference model, and its logarithmic loss was 0.4807, compared with the reference value of 0.6179 (Table 11). Thus, TabM outperformed the corresponding uninformative reference according to all four measures. Figure 9 presents the ROC and precision–recall curves across the complete range of classification thresholds.

Table 11. Performance of the TabM model on the grouped-random test dataset.

Measure	TabM	Reference Value	Preferred Direction
Average precision	0.6852	0.3071	Higher
AUROC	0.7982	0.5000	Higher
Brier score	0.1549	0.2133	Lower
Logarithmic loss	0.4807	0.6179	Lower

The risk ranking concentrated noncompliant inspections and recorded violations within the highest-risk portions of the test dataset (Table 12). At the nominal 10% capacity level, the ranking selected 422 inspections, of which 353 were noncompliant. This yielded 83.65% precision, 27.26% recall, and a lift of 2.72. The selected inspections contained 2284 of the 5356 recorded violations, corresponding to 42.64% violation coverage. As capacity increased from 10% to 30%, recall increased from 27.26% to 62.39%, and violation coverage increased from 42.64% to 73.92%, whereas precision decreased from 83.65% to 63.82%, and lift decreased from 2.72 to 2.08.

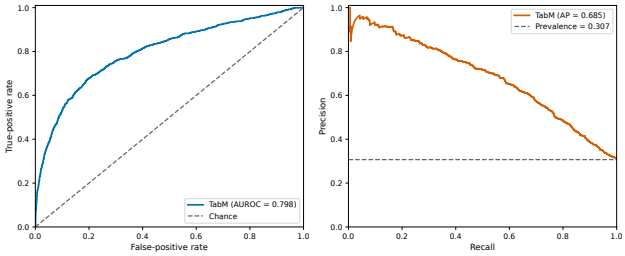


Figure 9. ROC and precision–recall curves for TabM on the test dataset under the grouped-random evaluation design.

Table 12. Capacity-constrained performance of TabM on the test dataset under the grouped-random evaluation design.

Capacity	Selected Inspections	Noncompliant Captured	Precision	Recall	Lift	Violations Captured	Violation Coverage
10%	422	353	83.65%	27.26%	2.72	2284	42.64%
20%	844	611	72.39%	47.18%	2.36	3308	61.76%
30%	1266	808	63.82%	62.39%	2.08	3959	73.92%

Figure 10 presents the cumulative gain across the complete TabM ranking. The curve remained above the random-ranking reference, demonstrating that TabM concentrated noncompliant inspections in the highest-ranked portion of the test dataset.

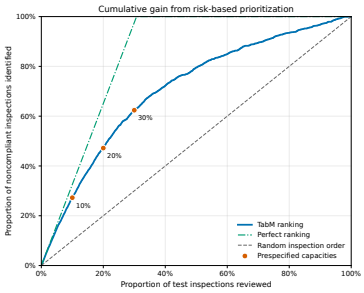


Figure 10. Cumulative gain from the TabM risk ranking on the grouped-random test dataset.

Figure 11 shows that Total_previous_inspections and Previous_number_of_violations had substantially greater permutation importance than the remaining predictors. Permuting these variables reduced average precision by 0.1171 and 0.1165, respectively.

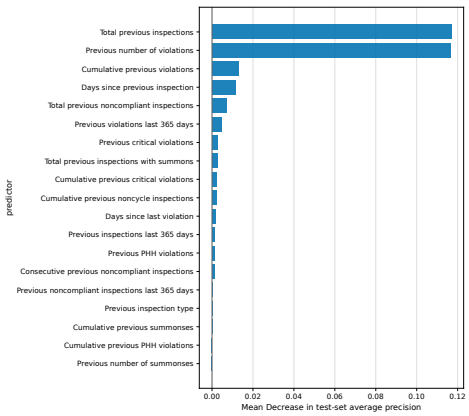


Figure 11. Permutation importance of the 19 predictors for TabM on the test dataset under the grouped random evaluation.

5.1.4. Comparison of Algorithms

Because all three algorithms were evaluated on the same 4217 test inspections, we compared their independent-test results directly. Table 13 compares the four test-performance measures, reporting small differences between the three algorithms with respect to the average precision, AUROC, Brier score, and logarithmic loss.

Table 13. Comparison of the three algorithms on the grouped-random test dataset.

Algorithm	Average Precision	AUROC	Brier Score	Logarithmic Loss
Logistic regression	0.6695	0.7864	0.1594	0.4920
XGBoost	0.6832	0.8003	0.1539	0.4780
TabM	0.6852	0.7982	0.1549	0.4807

The capacity-constrained results showed similarly small differences between the three algorithms with respect to precision, recall, lift, noncompliance captured, and violation captured (Table 14).

Overall, the nonlinear models produced modest improvements over logistic regression in discrimination and in the concentration of noncompliant inspections near the top of the ranking. The difference between XGBoost and TabM was much smaller. TabM attained the highest average precision and captured the most violations at 10% capacity. In contrast, XGBoost attained the highest AUROC, the lowest Brier score and logarithmic loss, and the strongest noncompliance-capture results at 20% and 30% capacity. Logistic regression generally produced lower discrimination and captured fewer noncompliant inspections, but its capacity-constrained violation coverage remained close to that of the two more complex algorithms. These comparisons describe the observed test-set results. Consequently, the small numerical advantages of XGBoost or TabM should not be interpreted as evidence that either algorithm has statistically superior generalization performance.

Table 14. Comparison of capacity-constrained performance on the grouped-random test dataset.

Capacity	Algorithm	Noncompliant Captured	Precision	Recall	Lift	Violations Captured	Violation Coverage
10%	Logistic regression	344	81.52%	26.56%	2.65	2267	42.33%
	XGBoost	353	83.65%	27.26%	2.72	2274	42.46%
	TabM	353	83.65%	27.26%	2.72	2284	42.64%
20%	Logistic regression	598	70.85%	46.18%	2.31	3309	61.78%
	XGBoost	614	72.75%	47.41%	2.37	3327	62.12%
	TabM	611	72.39%	47.18%	2.36	3308	61.76%
30%	Logistic regression	796	62.88%	61.47%	2.05	3965	74.03%
	XGBoost	815	64.38%	62.93%	2.10	3966	74.05%
	TabM	808	63.82%	62.39%	2.08	3959	73.92%

The permutation-importance analyses identified a consistent substantive pattern across the three algorithms (Figures 5, 8 and 11). All three models depended most strongly on Previous_number_of_violations and Total_previous_inspections. For logistic regression, permuting these predictors reduced test-set average precision by 0.2655 and 0.1354, respectively; the corresponding reductions for XGBoost were 0.1263 and 0.0641. TabM depended almost equally on the same two predictors, although their order was reversed: the mean reductions were 0.1171 for Total_previous_inspections and 0.1165 for Previous_number_of_violations. Importance declined for the three models sharply after these two predictors. These findings indicate that the extent of noncompliance at the immediately preceding inspection and the depth of the available inspection history provided the most consistently useful ranking information across the three model structures [27,34].

5.2. Chronological Evaluation Results

5.2.1. Logistic-Regression Results

The five-iteration temporal-validation procedure selected $C = 0.1$ by the prespecified criterion of mean validation average precision. At this value, the mean validation average precision was 0.3494. After model selection, we refitted the selected configuration using all 24,242 development inspections and evaluated it once on the 5421 temporally held-out test inspections, of which 890 (16.42%) were noncompliant. Table 15 reports the four prespecified test-performance measures and their reference values.

Table 15. Performance of the logistic-regression model on the held-out chronological test dataset.

Measure	Logistic Regression	Reference Value	Preferred Direction
Average precision	0.2645	0.1642	Higher
AUROC	0.6221	0.5000	Higher
Brier score	0.1390	0.1602	Lower
Logarithmic loss	0.4749	0.5065	Lower

The model's average precision exceeded the test-prevalence reference, and its AUROC exceeded the chance reference. Its Brier score and logarithmic loss were also lower than those of the constant reference that assigned the development-set noncompliance prevalence of 0.3159 to every test inspection.

Figure 12 shows the receiver operating characteristic and precision–recall curves. The receiver operating characteristic curve remained above the chance diagonal over almost the entire false-positive-rate range. In contrast, the precision–recall curve remained above the test-prevalence reference over most of the recall range. Precision was highest at low recall and generally declined as recall increased.

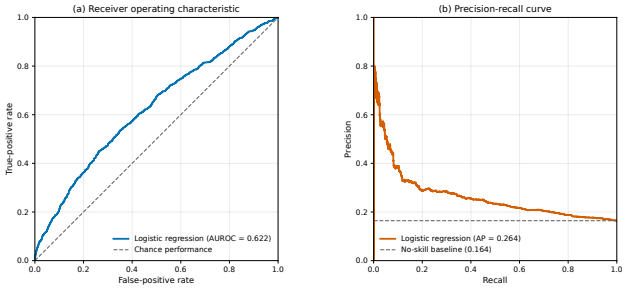


Figure 12. Discrimination of the logistic-regression model on the chronological test dataset.

The risk ranking concentrated noncompliant inspections and recorded violations within the highest-ranked portions of the test dataset (Table 16). At the nominal 10% capacity level, the ranking selected 543 inspections, of which 167 were noncompliant. This yielded 30.76% precision, 18.76% recall, and a lift of 1.87. The selected inspections contained 662 of the 2753 recorded violations, corresponding to 24.05% violation coverage. As capacity increased from 10% to 30%, recall increased from 18.76% to 45.28%, and violation coverage increased from 24.05% to 52.20%, whereas precision decreased from 30.76% to 24.77%, and lift decreased from 1.87 to 1.51.

Table 16. Capacity-constrained performance of the logistic-regression risk ranking on the chronological test dataset.

Capacity	Inspections Selected	Noncompliant Inspections Captured	Precision	Recall	Lift	Violations Captured	Violation Coverage
10%	543	167	30.76%	18.76%	1.87	662	24.05%
20%	1085	298	27.47%	33.48%	1.67	1115	40.50%
30%	1627	403	24.77%	45.28%	1.51	1437	52.20%

Figure 13 extends the capacity analysis across the complete test ranking. At each prespecified capacity level, the observed cumulative gain exceeded the random-order reference.

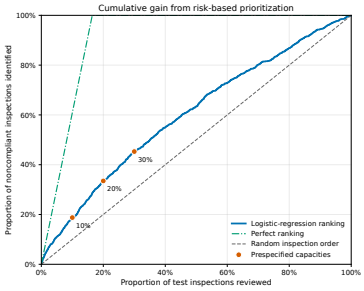


Figure 13. Cumulative gain from the logistic-regression risk ranking on the chronological test dataset.

The largest mean decrease in test-set average precision was 0.0635, obtained when we permuted Total_previous_noncompliant_inspections (Figure 14). For the predictors with negative mean decrease in test-set average precision, the average precision was higher after permutation on average.

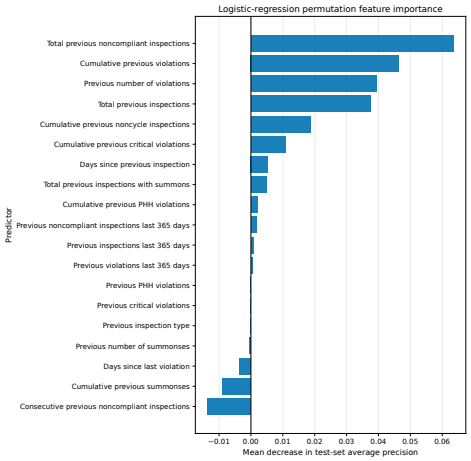


Figure 14. Permutation-based feature importance for the logistic-regression model on the chronological test dataset.

5.2.2. XGBoost Chronological Results

The selected XGBoost configuration achieved a mean validation average precision of 0.3507. We refitted the selected configuration using all 24,242 development inspections recorded from 2017 through 2023. We then evaluated the fitted model on the 5421 temporally held-out inspections conducted from January 2024 through March 2026. The test dataset contained 890 noncompliant inspections, corresponding to a noncompliance prevalence of 16.42%. Table 17 reports the four prespecified test-performance measures and their reference values.

On the chronological test dataset, XGBoost achieved an average precision of 0.2893 and an AUROC of 0.6490. Average precision exceeded the no-skill reference of 0.1642, which equals test-set noncompliance prevalence, while AUROC exceeded the chance reference of 0.5000. Thus, XGBoost retained an ability to rank future noncompliant inspections above compliant inspections, although its discrimination remained moderate. The model produced a Brier score of 0.1308 and a logarithmic loss of 0.4258. Assigning the development-set noncompliance prevalence of 31.59% to every test inspection produced corresponding reference values of 0.1602 and 0.5065. XGBoost therefore outperformed this prespecified constant-probability reference on both measures.

Table 17. Performance of XGBoost on the chronological test dataset.

Measure	XGBoost	Reference	Preferred Direction
Average precision	0.2893	0.1642	Higher
AUROC	0.6490	0.5000	Higher
Brier score	0.1308	0.1602	Lower
Logarithmic loss	0.4258	0.5065	Lower

Figure 15 presents the receiver operating characteristic and precision–recall curves constructed from the test outcomes and XGBoost-predicted probabilities. The AUROC summarizes the model's ranking across the complete range of classification thresholds, whereas average precision summarizes the precision–recall relationship. Neither measure represents performance at a single classification threshold.

The chronological XGBoost ranking concentrated noncompliant inspections and recorded violations within the highest-ranked portions of the test dataset (Table 18). Figure 16 shows cumulative gain across the complete test ranking. At the prespecified 10%, 20%, and 30% capacity levels, XGBoost identified 22.02%, 36.52%, and 46.29% of all noncompliant test inspections, respectively. Each value exceeded the proportion expected under random inspection order at the corresponding capacity. However, the distance between the XGBoost and perfect-ranking curves indicates that substantial noncompliance remained outside the highest-ranked inspections.

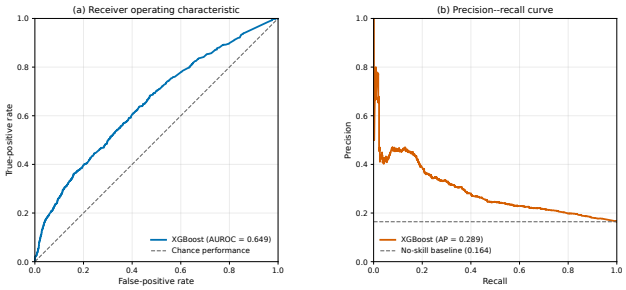


Figure 15. Discrimination of XGBoost on the chronological test dataset.

Table 18. Capacity-constrained performance of the XGBoost risk ranking on the chronological test dataset.

Capacity	Inspections Selected	Noncompliant Identified	Precision	Recall	Lift	Violations Identified	Violation Coverage
10%	543	196	36.10%	22.02%	2.20	720	26.15%
20%	1085	325	29.95%	36.52%	1.82	1104	40.10%
30%	1627	412	25.32%	46.29%	1.54	1383	50.24%

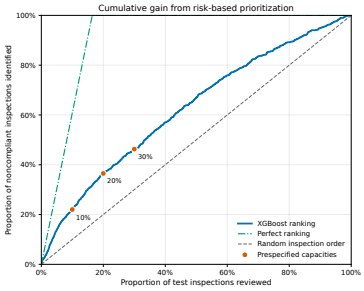


Figure 16. Cumulative gain from XGBoost risk-based prioritization on the chronological test dataset.

Permutation importance showed that XGBoost's test-set ranking depended most strongly on temporal and cumulative violation-history predictors. Permuting Days_since_last_violation produced the largest mean reduction in test-set average precision, at 0.0249. The next-largest reductions were 0.0190 for Cumulative_previous_violations, and 0.0154 for Previous_number_of_violations (Figure 17).

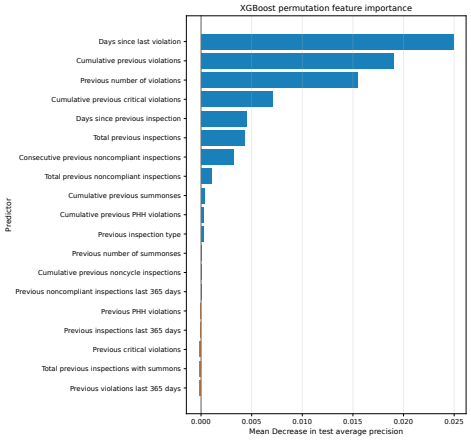


Figure 17. Permutation importance of the 19 historical predictors for XGBoost on the chronological test dataset.

5.2.3. TabM Chronological Results

The temporal-validation procedure selected 17 training epochs for the prespecified TabM configuration. At this epoch, TabM achieved a mean validation average precision of 0.3535. We fitted the final TabM model for 17 epochs using all 24,242 development inspections recorded from 2017 through 2023. We then evaluated it on the 5421 temporally held-out inspections conducted from January 2024 through March 2026. The test dataset contained 890 noncompliant inspections, corresponding to a noncompliance prevalence of 16.42%, compared with 31.59% in the development dataset.

TabM achieved a test average precision of 0.2939 and an AUROC of 0.6467 (Table 19). Its average precision exceeded the no-skill reference of 0.1642, which equals the test-set noncompliance prevalence, while its AUROC exceeded the chance reference of 0.5000. TabM produced a Brier score of 0.1322 and a logarithmic loss of 0.4355. The prespecified constant-probability reference, which assigned the 31.59% development prevalence to every test inspection, produced a Brier score of 0.1602 and a logarithmic loss of 0.5065. TabM therefore outperformed this reference on both measures.

Figure 18 presents the receiver operating characteristic and precision–recall curves obtained from the test outcomes and TabM-predicted probabilities. Both curves summarize ranking performance across the complete range of classification thresholds rather than performance at a single threshold.

Table 19. Performance of TabM on the chronological test dataset.

Measure	TabM	Reference	Preferred Direction
Average precision	0.2939	0.1642	Higher
AUROC	0.6467	0.5000	Higher
Brier score	0.1322	0.1602	Lower
Logarithmic loss	0.4355	0.5065	Lower

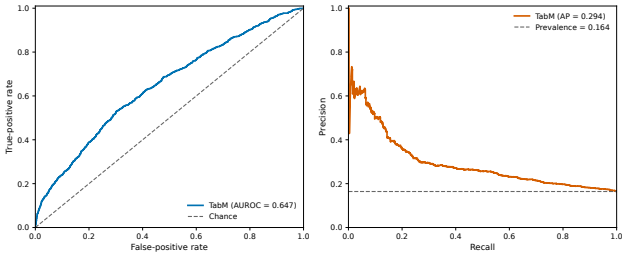


Figure 18. Discrimination of TabM on the chronological test dataset.

The TabM risk ranking concentrated noncompliant inspections and recorded violations within the highest-ranked portions of the test dataset (Table 20). Figure 19 presents cumulative gain across the complete test ranking. By inspecting the highest-ranked 10%, 20%, and 30% of inspections, TabM identified 21.24%, 34.72%, and 47.87% of all noncompliant test inspections, respectively. Each value exceeded the corresponding 10%, 20%, or 30% expected under random inspection order.

Table 20. Capacity-constrained performance of the TabM risk ranking on the chronological test dataset.

Capacity	Inspections Selected	Noncompliant Identified	Precision	Recall	Lift	Violations Identified	Violation Coverage
10%	543	189	34.81%	21.24%	2.12	722	26.23%
20%	1085	309	28.48%	34.72%	1.73	1101	39.99%
30%	1627	426	26.18%	47.87%	1.59	1463	53.14%

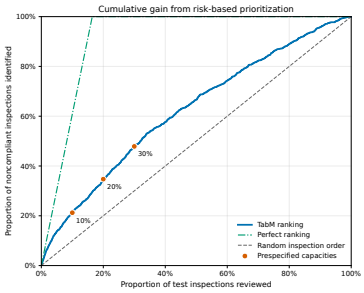


Figure 19. Cumulative gain from TabM risk-based prioritization on the chronological test dataset.

Permutation importance showed that TabM's test-set ranking depended most strongly on cumulative inspection- and violation-history predictors (Figure 20). Permuting Cumulative_previous_violations produced the largest mean decrease in test average precision, at 0.0407. The next-largest decreases were 0.0346 for Total_previous_inspections, 0.0253 for Total_previous_noncompliant_inspections, and 0.0144 for Cumulative_previous_noncycle_inspections.

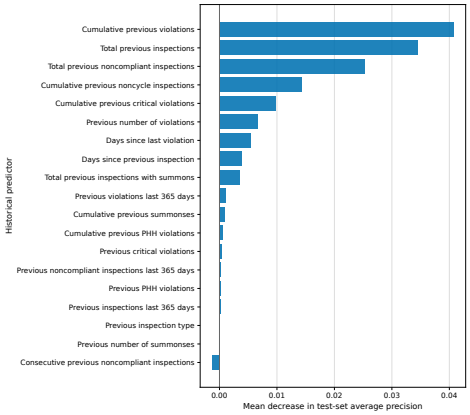


Figure 20. Permutation importance of the 19 historical predictors for TabM on the chronological test dataset.

5.2.4. Comparison of Algorithms Under Chronological Evaluation

Because we evaluated all three algorithms on the same 5421 temporally held-out inspections, differences among their test measures did not arise from differences in test-set composition. The mean validation average precision values used for model selection differed by only 0.0041, ranging from 0.3494 for logistic regression to 0.3535 for TabM (Table 21). TabM produced the highest mean validation average precision.

On the temporally held-out test dataset, the two nonlinear algorithms produced more favorable observed values than logistic regression for all four measures. TabM achieved the highest average precision (0.2939), whereas XGBoost achieved the highest AUROC (0.6490) and the lowest Brier score (0.1308) and logarithmic loss (0.4258). The differences between TabM and XGBoost were small: TabM's average precision was 0.0046 higher, while XGBoost's AUROC was 0.0023 higher, and its Brier score and logarithmic loss were 0.0014 and 0.0097 lower, respectively. Thus, neither algorithm produced the best observed value for every test measure.

Table 21. Mean validation average precision and test performance of the three algorithms under chronological evaluation.

Algorithm	Mean Validation Average Precision	Test Average Precision	Test AUROC	Test Brier Score	Test Logarithmic Loss
Logistic regression	0.3494	0.2645	0.6221	0.1390	0.4749
XGBoost	0.3507	0.2893	0.6490	0.1308	0.4258
TabM	0.3535	0.2939	0.6467	0.1322	0.4355

The algorithms also differed across the prespecified inspection-capacity levels (Table 22). Because each algorithm selected the same number of inspections at a given capacity, the algorithm that identified the most noncompliant inspections also achieved the highest precision, recall, and lift at that capacity. XGBoost achieved these highest values at 10% and 20% capacity, identifying 196 and 325 noncompliant inspections, respectively. At 30% capacity, TabM produced the highest value for every reported capacity-constrained measure.

The ordering differed for recorded violations. TabM identified the most violations at 10% capacity (722), logistic regression identified the most at 20% capacity (1115), and TabM identified the most at 30% capacity (1463). We trained the algorithms to predict the binary outcome Any_violation, which assigns the same positive label to an inspection with one violation and an inspection with several violations. Consequently, a ranking that identified more noncompliant inspections did not necessarily identify more recorded violations.

Table 22. Capacity-constrained performance of the three algorithms under chronological evaluation.

Capacity	Algorithm	Noncompliant Identified	Precision	Recall	Lift	Violations Identified	Violation Coverage
10%	Logistic regression	167	30.76%	18.76%	1.87	662	24.05%
	XGBoost	196	36.10%	22.02%	2.20	720	26.15%
	TabM	189	34.81%	21.24%	2.12	722	26.23%
20%	Logistic regression	298	27.47%	33.48%	1.67	1115	40.50%
	XGBoost	325	29.95%	36.52%	1.82	1104	40.10%
	TabM	309	28.48%	34.72%	1.73	1101	39.99%
30%	Logistic regression	403	24.77%	45.28%	1.51	1437	52.20%
	XGBoost	412	25.32%	46.29%	1.54	1383	50.24%
	TabM	426	26.18%	47.87%	1.59	1463	53.14%

The permutation analyses showed both a common pattern and model-specific differences. Cumulative_previous_violations ranked among the two largest mean decreases in test-set average precision for every algorithm. However, the largest mean decrease occurred for Total_previous_noncompliant_inspections with logistic regression, Days_since_last_violation with XGBoost, and Cumulative_previous_violations with TabM.

5.3. Comparison of Evaluation Designs

The grouped-random test dataset contained 4217 inspections, of which 1295 were noncompliant, yielding a noncompliance prevalence of 30.71%. The chronological test dataset contained 5421 inspections, of which 890 were noncompliant, yielding a prevalence of 16.42%. The chronological test dataset was therefore larger but contained fewer noncompliant inspections, and its prevalence was 14.29 percentage points lower.

Within each design, all three algorithms outperformed the corresponding reference on each overall test measure (Table 23). However, all three algorithms produced lower average precision and AUROC under chronological evaluation. The models therefore discriminated less effectively between noncompliant and compliant inspections in the later test dataset than in the grouped-random test dataset.

Table 23. Overall test performance under the grouped-random and chronological evaluation designs.

Evaluation Design	Algorithm	Average Precision	AUROC	Brier Score	Logarithmic Loss
Grouped random	Logistic regression	0.6695	0.7864	0.1594	0.4920
	XGBoost	0.6832	0.8003	0.1539	0.4780
	TabM	0.6852	0.7982	0.1549	0.4807
	Constant-score reference	0.3071	0.5000	0.2133	0.6179
Chronological	Logistic regression	0.2645	0.6221	0.1390	0.4749
	XGBoost	0.2893	0.6490	0.1308	0.4258
	TabM	0.2939	0.6467	0.1322	0.4355
	Constant-score reference	0.1642	0.5000	0.1602	0.5065

The raw Brier scores and logarithmic losses were numerically lower under the chronological design, but the corresponding constant-score references were also lower. The capacity-constrained results showed the same overall pattern (Table 24). At every capacity level and for every algorithm, chronological evaluation produced lower recall, lift, and violation coverage than grouped-random evaluation. Nevertheless, every chronological lift value exceeded 1, indicating that the later-period rankings still concentrated noncompliant inspections above the random-order reference.

Table 24. Capacity-constrained test performance under the grouped-random and chronological evaluation designs.

Capacity	Algorithm	Recall		Lift		Violation Coverage	
		Grouped Random	Chronological	Grouped Random	Chronological	Grouped Random	Chronological
10%	Logistic regression	26.56%	18.76%	2.65	1.87	42.33%	24.05%
	XGBoost	27.26%	22.02%	2.72	2.20	42.46%	26.15%
	TabM	27.26%	21.24%	2.72	2.12	42.64%	26.23%
20%	Logistic regression	46.18%	33.48%	2.31	1.67	61.78%	40.50%
	XGBoost	47.41%	36.52%	2.37	1.82	62.12%	40.10%
	TabM	47.18%	34.72%	2.36	1.73	61.76%	39.99%
30%	Logistic regression	61.47%	45.28%	2.05	1.51	74.03%	52.20%
	XGBoost	62.93%	46.29%	2.10	1.54	74.05%	50.24%
	TabM	62.39%	47.87%	2.08	1.59	73.92%	53.14%

5.4. Rule-Based Prioritization

All four history rules produced lift values above 1 under both evaluation designs and at every capacity level, indicating better concentration of noncompliant inspections than random ordering (Table 25). Under grouped-random evaluation, the previous violation count was the strongest rule for every reported measure and capacity level. The strongest chronological rule depended on the capacity level and outcome.

The fitted models generally improved on the strongest rules, but the gains were modest. Under grouped-random evaluation, every model exceeded the strongest rule in recall at each capacity level; all of them also exceeded it in violation coverage except logistic regression at 10% capacity. Under chronological evaluation, XGBoost and TabM exceeded the highest rule-based recall at every capacity level, whereas logistic regression did not. All three models exceeded the highest rule-based violation coverage at every capacity level. Thus, combining the historical predictors provided incremental improvements over prioritization based on a single history measure.

Table 25. Capacity-constrained performance of the four rule-based comparators under grouped-random and chronological evaluation.

Capacity	Rule	Grouped-Random			Chronological		
		Recall	Lift	Violation Coverage	Recall	Lift	Violation Coverage
10%	Previous inspection outcome	16.50%	1.65	18.60%	17.26%	1.72	18.83%
	Previous violation count	26.12%	2.61	42.42%	20.31%	2.03	23.82%
	Cumulative previous violations	13.17%	1.32	14.83%	19.58%	1.95	22.91%
	Violation recency	17.45%	1.74	23.76%	16.77%	1.67	20.08%
20%	Previous inspection outcome	33.01%	1.65	37.21%	30.90%	1.54	33.26%
	Previous violation count	45.51%	2.27	60.44%	30.90%	1.54	33.26%
	Cumulative previous violations	22.69%	1.13	22.50%	32.85%	1.64	36.65%
	Violation recency	34.63%	1.73	41.91%	33.93%	1.70	36.40%
30%	Previous inspection outcome	49.51%	1.65	55.81%	39.54%	1.32	41.61%
	Previous violation count	60.43%	2.01	73.19%	39.54%	1.32	41.61%
	Cumulative previous violations	32.50%	1.08	30.98%	42.85%	1.43	47.40%
	Violation recency	50.73%	1.69	58.31%	45.39%	1.51	47.62%

6. Discussion

6.1. Main Findings and Interpretation

This study found that a cooling-tower system's recorded inspection history contained useful information about noncompliance at its next inspection. Each algorithm produced an informative retrospective ranking under both evaluation designs. The findings therefore

demonstrate predictive value within the evaluated data, but they do not establish that deploying a model would improve inspection outcomes.

The chronological results provide the more relevant guide to expected performance when prioritizing future inspections, because the models were trained on earlier inspections and tested on later ones [9]. Grouped-random evaluation assessed transfer to systems excluded from model development. All three models had lower average precision and AUROC under chronological evaluation. Noncompliance prevalence was also lower in the chronological test set (16.42%) than in the grouped-random test set (30.71%).

The raw Brier scores and logarithmic losses were numerically lower under the chronological design, but the corresponding constant-score references were also lower. Because the magnitudes of these probabilistic scores depend on the outcome distribution, the lower raw values do not by themselves establish better performance [45,46].

We treated the cross-design comparisons as descriptive. The designs used different test inspections and differed simultaneously in calendar period, system composition, sample size, and noncompliance prevalence. The experiment therefore cannot isolate the contributions of temporal change, prevalence change, or other differences between the test populations.

Differences among the algorithms were small relative to the difference between evaluation designs, and no algorithm showed a decisive overall advantage. The nonlinear models performed somewhat better on some measures, but logistic regression also produced informative rankings. A model selected for prospective use should therefore balance predictive performance with transparency, implementation effort, and ease of updating.

The timing results add another practical consideration. Model selection and final training together took 12.66 min for TabM, 4.09 min for XGBoost, and 0.26 min for logistic regression under grouped-random evaluation. The corresponding chronological times were 22.84, 1.70, and 0.10 min (Table 2). On this CPU system, TabM took longer than the other implemented procedures, without a decisive overall predictive advantage. The time required for repeated model selection and training should therefore be considered alongside predictive performance when choosing a model for routine use.

The permutation analyses indicated that the fitted models relied mainly on previous and cumulative violation history, the extent of the recorded inspection history, and the timing of earlier inspections or violations. This finding is consistent with research showing that historical regulatory records can support inspection targeting in other environmental domains [13]. However, permutation importance describes a fitted model's reliance on a predictor; it does not establish that the predictor causes later noncompliance [34].

We tested whether deleting same-day Non-Cycle records affected the results by combining their violations and summonses with the Cycle records and refitting the models. The sensitivity analysis showed only small changes in overall model performance (Supplementary Table S1).

The models could help regulators prioritize repeat inspections using recorded inspection history. They predict any recorded violation, not violation count or severity, *Legionella* contamination, or public-health harm. A low predicted probability does not establish compliance or safety. The rankings should support inspectors' judgment and should not replace required inspections, water testing, or evidence needed for enforcement. Prospective evaluation is needed before operational use.

6.2. Limitations

We developed and evaluated the models using administrative records from a single municipal regulatory program. The grouped-random and chronological designs assessed system-level and temporal generalization separately: the former used overlapping calendar

periods, whereas the latter included systems observed during model development. Neither design jointly evaluated temporal transfer and transfer to previously unseen systems, and neither provided external validation in another jurisdiction. The estimates therefore remain specific to the observed New York City setting.

The outcome was recorded regulatory noncompliance, which may depend on inspection scope and type, documentation, regulatory definitions, inspector practice, and the selection of systems for non-routine inspection. It is therefore an imperfect measure of a cooling tower's underlying condition, and an inspection without a recorded violation does not establish that no deficiency existed.

The models used only information from earlier inspections and excluded each system's first recorded inspection. They may therefore omit other information useful for ranking repeat inspections and cannot score newly registered or previously uninspected systems.

The algorithm comparisons were descriptive. We evaluated one prespecified TabM architecture while tuning logistic regression and XGBoost. The findings therefore compare the implemented modeling procedures, not the best attainable performance of the three algorithm classes. The 48-configuration XGBoost search may have missed better settings. A larger search could improve XGBoost performance and change the comparisons between algorithms.

Each running time was measured in one subsequent rerun on one computer. The times depend on hardware, settings for parallel processing, and background activity. In addition, the models used different selection procedures and numbers of candidate configurations. The timing comparison is therefore specific to the procedures and computer setup used here.

Some predictors provide similar information (Supplementary Figure S1). When one predictor is shuffled, the model may still obtain that information from other predictors. A low permutation-importance score therefore does not necessarily mean that the predictor is unhelpful. Our correlation analysis helps identify this overlap, but it does not remove it. It may also miss more complex relationships, including cases where several predictors together provide the same information as another predictor. Therefore, the importance rankings should not be treated as precise measures of how much each predictor contributes independently of the others.

The capacity analysis evaluated how effectively the models ranked the inspections in the test data. It did not evaluate the models within an actual inspection program or account for practical constraints such as inspection requirements, inspector workload, travel time, and system operating schedules. Therefore, the results show retrospective prioritization performance, not how model-assisted inspection planning would perform in practice.

6.3. Future Work

Because the models produced only modest gains over simple history rules, a short prospective pilot should compare model-assisted planning with current practice and a rule based on previous violations. Each approach should use the same number of inspections. The pilot should test which approach finds more violations.

Future models should align the outcome and predictors with the intended regulatory objective. Possible outcomes include any violation, violation count, and separate outcomes for public-health hazard (PHH) and Critical violations. Researchers should test whether adding pre-inspection building, system, maintenance, water-quality, environmental, and regulatory information improves performance on later inspections. Predicting first inspections would require a separate model, or an expanded model designed for systems without inspection histories, using only pre-inspection information.

Given the lower performance on later inspections, future studies should compare models trained once with models periodically retrained on newer records. Both approaches

should be tested on the same later inspections. Validation should also test later inspections of systems excluded from model development, assessing temporal and system-level transfer together.

Future external validation should evaluate performance in other jurisdictions while accounting for local differences in data quality, cooling-tower populations, and noncompliance prevalence.

7. Conclusions

Using longitudinal New York City cooling-tower inspection records, we found that inspection history provided useful information for predicting recorded noncompliance at the next inspection and supported retrospective prioritization. All four simple history rules concentrated noncompliant inspections more effectively than random ordering at the examined capacity levels. At the same time, the fitted models generally provided only modest additional gains by combining multiple history predictors. Logistic regression, XGBoost, and TabM achieved broadly similar predictive performance, but all three performed less effectively on chronologically later inspections than on the grouped-random test inspections. The observed differences between the evaluation designs were larger than those among the algorithms, highlighting the importance of temporal evaluation. These findings support prospective evaluation of model-assisted prioritization, but they do not establish its operational effectiveness. The models predict recorded regulatory noncompliance, not public-health risk.

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/informatics13110177/s1, Figure S1 shows pairwise Spearman rank correlation matrices for the 19 historical predictors in the grouped-random and chronological development datasets; Table S1 reports sensitivity to same-day record handling.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The raw data are publicly available within NYC Cooling Tower System Inspection Results [8].

Acknowledgments: The author gratefully acknowledges the American University of Sharjah and the German Jordanian University for providing supportive academic environments that foster research and scholarly development. The views and opinions expressed in this article are solely those of the author and do not necessarily reflect the views or positions of either the American University of Sharjah or the German Jordanian University.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Mouchtouri, V.A.; Goutziana, G.; Kremastinou, J.; Hadjichristodoulou, C. Legionella Species Colonization in Cooling Towers: Risk Factors and Assessment of Control Measures. *Am. J. Infect. Control* **2010**, *38*, 50–55. https://doi.org/10.1016/j.ajic.2009.04.285.
2. Lapierre, P.; Nazarian, E.; Zhu, Y.; Wroblewski, D.; Saylors, A.; Passaretti, T.; Hughes, S.; Tran, A.; Lin, Y.; Kornblum, J.; et al. Legionnaires' Disease Outbreak Caused by Endemic Strain of *Legion. Pneumophila*, New York, New York, USA, 2015. *Emerg. Infect. Dis.* **2017**, *23*, 1784–1791. https://doi.org/10.3201/eid2311.170308.
3. Campaña, M.; del Hoyo, R.; Monleón-Getino, A.; Checa, J. Predicting Legionella Contamination in Cooling Towers and Evaporative Condensers from Microbiological and Physicochemical Parameters. *Int. J. Hyg. Environ. Health* **2023**, *248*, 114117. https://doi.org/10.1016/j.ijheh.2023.114117.
4. New York City Department of Health and Mental Hygiene. Available online: https://www.nyc.gov (accessed on 20 July 2026).

5. Wong, K.K.; Segura, T.; Mein, G.; Lu, J.; Hannapel, E.J.; Kunz, J.M.; Ritter, T.; Smith, J.C.; Todeschini, A.; Nugen, F.; et al. Automated Cooling Tower Detection through Deep Learning for Legionnaires' Disease Outbreak Investigations: A Model Development and Validation Study. *Lancet Digit. Health* **2024**, *6*, e500–e506. https://doi.org/10.1016/S2589-7500(24)00094-3.
6. Oldroyd, R.A.; Morris, M.A.; Birkin, M. Predicting Food Safety Compliance for Informed Food Outlet Inspections: A Machine Learning Approach. *Int. J. Environ. Res. Public Health* **2021**, *18*, 12635. https://doi.org/10.3390/ijerph182312635.
7. Robb, K.A.; Diaz Amigo, N.; Marcoux, A.; McAteer, M.; de Jong, J. Using Integrated City Data and Machine Learning to Identify and Intervene Early on Housing-Related Public Health Problems. *J. Public Health Manag. Pract.* **2022**, *28*, E497–E505. https://doi.org/10.1097/PHH.0000000000001343.
8. New York City Department of Health and Mental Hygiene. NYC Cooling Tower System Inspection Results. Dataset Last Updated 17 April 2026. 2026. Available online: https://data.cityofnewyork.us/Health/NYC-Cooling-Tower-System-Inspection-Results/f9wb-g8mb (accessed on 20 July 2026).
9. Roberts, D.R.; Bahn, V.; Ciuti, S.; Boyce, M.S.; Elith, J.; Guillera-Arroita, G.; Hauenstein, S.; Lahoz-Monfort, J.J.; Schröder, B.; Thuiller, W.; et al. Cross-Validation Strategies for Data with Temporal, Spatial, Hierarchical, or Phylogenetic Structure. *Ecography* **2017**, *40*, 913–929. https://doi.org/10.1111/ecog.02881.
10. Collins, G.S.; Moons, K.G.M.; Dhiman, P.; Riley, R.D.; Beam, A.L.; Van Calster, B.; Ghassemi, M.; Liu, X.; Reitsma, J.B.; van Smeden, M.; et al. TRIPOD+AI Statement: Updated Guidance for Reporting Clinical Prediction Models That Use Regression or Machine Learning Methods. *BMJ* **2024**, *385*, e078378. https://doi.org/10.1136/bmj-2023-078378. Correction in *BMJ* **2024**, *385*, q902.
11. Crook, B.; Willerton, L.; Smith, D.; Wilson, L.; Poran, V.; Helps, J.; McDermott, P. Legionella Risk in Evaporative Cooling Systems and Underlying Causes of Associated Breaches in Health and Safety Compliance. *Int. J. Hyg. Environ. Health* **2020**, *224*, 113425. https://doi.org/10.1016/j.ijheh.2019.113425.
12. Girolamini, L.; Brattich, E.; Marino, F.; Pascale, M.R.; Mazzotta, M.; Spiteri, S.; Derelitto, C.; Tositti, L.; Cristino, S. Cooling Towers Influence in an Urban Environment: A Predictive Model to Control and Prevent *Legion. Risk Legion. Events. Build. Environ.* **2023**, *228*, 109891. https://doi.org/10.1016/j.buildenv.2022.109891.
13. Hino, M.; Benami, E.; Brooks, N. Machine Learning for Environmental Monitoring. *Nat. Sustain.* **2018**, *1*, 583–588. https://doi.org/10.1038/s41893-018-0142-9.
14. Zhou, Q.; Qu, S.; Wang, Q.; She, Y.; Yu, Y.; Bi, J. Sliding Window-Based Machine Learning for Environmental Inspection Resource Allocation. *Environ. Sci. Technol.* **2023**, *57*, 16743–16754. https://doi.org/10.1021/acs.est.3c05088.
15. Manu, J.; Adjei, F. Predictive Compliance Modeling for Resource Conservation and Recovery Act (RCRA) Hazardous Waste Generators: A Data-Driven Approach for Inspection Targeting. *J. Eng. Res. Rep.* **2026**, *28*, 411–419. https://doi.org/10.9734/jerr/2026/v28i41872.
16. Kang, J.S.; Kuznetsova, P.; Luca, M.; Choi, Y. Where Not to Eat? Improving Public Policy by Predicting Hygiene Inspections Using Online Reviews. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, Seattle, WA, USA, 18–21 October 2013; pp. 1443–1448.
17. Altenburger, K.M.; Ho, D.E. Is Yelp Actually Cleaning Up the Restaurant Industry? A Re-Analysis on the Relative Usefulness of Consumer Reviews. In *Proceedings of the 2019 World Wide Web Conference*; Association for Computing Machinery: New York, NY, USA, 2019; pp. 2543–2550. https://doi.org/10.1145/3308558.3313683.
18. Sadilek, A.; Caty, S.; DiPrete, L.; Mansour, R.; Schenk, Tom, J.; Bergtholdt, M.; Jha, A.; Ramaswami, P.; Gabrilovich, E. Machine-Learned Epidemiology: Real-Time Detection of Foodborne Illness at Scale. *Npj Digit. Med.* **2018**, *1*, 36. https://doi.org/10.1038/s41746-018-0045-1.
19. Schenk, Jr., T.; Leynes, G.; Solanki, A.; Collins, S.; Smart, G.; Albright, B.; Crippin, D. Forecasting Restaurants with Critical Violations in Chicago. City of Chicago GitHub Repository. Technical Report and Retrospective Evaluation. 2015. Available online: https://github.com/Chicago/food-inspections-evaluation/blob/master/REPORTS/forecasting-restaurants-with-critical-violations-in-Chicago.Rmd (accessed on 1 August 2026).
20. Johnson, M.S.; Levine, D.I.; Toffel, M.W. Improving Regulatory Effectiveness through Better Targeting: Evidence from OSHA. *Am. Econ. J. Appl. Econ.* **2023**, *15*, 30–67. https://doi.org/10.1257/app.20200659.
21. Rachman, A.; Ratnayake, R.M.C. Machine Learning Approach for Risk-Based Inspection Screening Assessment. *Reliab. Eng. Syst. Saf.* **2019**, *185*, 518–532. https://doi.org/10.1016/j.ress.2019.02.008.
22. Tsou, M.C. A Machine Learning-Based Model for Predicting High Deficiency Risk Ships in Port State Control: A Case Study of the Port of Singapore. *J. Mar. Sci. Eng.* **2025**, *13*, 1485. https://doi.org/10.3390/jmse13081485.
23. McBride, K.; Aavik, G.; Toots, M.; Kalvet, T.; Krimmer, R. How Does Open Government Data Driven Co-Creation Occur? Six Factors and a "Perfect Storm"; Insights from Chicago's Food Inspection Forecasting Model. *Gov. Inf. Q.* **2019**, *36*, 88–97. https://doi.org/10.1016/j.giq.2018.11.006.
24. Kannan, V.; Shapiro, M.A.; Bilgic, M. Hindsight Analysis of the Chicago Food Inspection Forecasting Model. In Proceedings of the AAAI Fall Symposium on Artificial Intelligence in Government and Public Sector, Arlington, VA, USA, 7–8 November 2019. https://doi.org/10.48550/arXiv.1910.04906.

25. Singh, S.; Shah, B.; Kanich, C.; Kash, I.A. Fair Decision-Making for Food Inspections. In *Proceedings of the 2nd ACM Conference on Equity and Access in Algorithms, Mechanisms, and Optimization*; Association for Computing Machinery: New York, NY, USA, 2022. https://doi.org/10.1145/3551624.3555289.
26. Cawley, G.C.; Talbot, N.L.C. On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation. *J. Mach. Learn. Res.* **2010**, *11*, 2079–2107.
27. Scikit-Learn Developers. Version 1.6 Documentation. Available online: https://scikit-learn.org (accessed on 30 July 2026).
28. Hastie, T.; Tibshirani, R.; Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed.; Springer: New York, NY, USA, 2009. https://doi.org/10.1007/978-0-387-84858-7.
29. Sokolova, M.; Lapalme, G. A Systematic Analysis of Performance Measures for Classification Tasks. *Inf. Process. Manag.* **2009**, *45*, 427–437. https://doi.org/10.1016/j.ipm.2009.03.002.
30. Boyd, K.; Eng, K.H.; Page, C.D. Area under the Precision–Recall Curve: Point Estimates and Confidence Intervals. In *Proceedings of the Machine Learning and Knowledge Discovery in Databases*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2013; Volume 8190, pp. 451–466. https://doi.org/10.1007/978-3-642-40994-3_29.
31. Saito, T.; Rehmsmeier, M. The Precision–Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. *PLoS ONE* **2015**, *10*, e0118432. https://doi.org/10.1371/journal.pone.0118432.
32. Fawcett, T. An Introduction to ROC Analysis. *Pattern Recognit. Lett.* **2006**, *27*, 861–874. https://doi.org/10.1016/j.patrec.2005.10.010.
33. Brier, G.W. Verification of Forecasts Expressed in Terms of Probability. *Mon. Weather Rev.* **1950**, *78*, 1–3. https://doi.org/10.1175/1520-0493(1950)078<0001:VOFEIT>2.0.CO;2.
34. Fisher, A.; Rudin, C.; Dominici, F. All Models Are Wrong, but Many Are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously. *J. Mach. Learn. Res.* **2019**, *20*, 1–81.
35. Cox, D.R. The Regression Analysis of Binary Sequences. *J. R. Stat. Soc. Ser. B* **1958**, *20*, 215–232. https://doi.org/10.1111/j.2517-6161.1958.tb00292.x.
36. le Cessie, S.; van Houwelingen, J.C. Ridge Estimators in Logistic Regression. *J. R. Stat. Soc. Ser. C* **1992**, *41*, 191–201. https://doi.org/10.2307/2347628.
37. Liu, D.C.; Nocedal, J. On the Limited Memory BFGS Method for Large Scale Optimization. *Math. Program.* **1989**, *45*, 503–528. https://doi.org/10.1007/BF01589116.
38. Chen, T.; Guestrin, C. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 13–17 August 2016; KDD '16; pp. 785–794. https://doi.org/10.1145/2939672.2939785.
39. XGBoost Developers. XGBoost 3.2 Documentation. Available online: https://xgboost.readthedocs.io (accessed on 26 July 2026).
40. Bergstra, J.; Bengio, Y. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.* **2012**, *13*, 281–305.
41. Gorishniy, Y.; Kotelnikov, A.; Babenko, A. TabM: Advancing Tabular Deep Learning with Parameter-Efficient Ensembling. In Proceedings of the Thirteenth International Conference on Learning Representations, Singapore, 24–28 April 2025.
42. Gorishniy, Y.; Kotelnikov, A.; Babenko, A. TabM: Official PyTorch Implementation. Version 0.0.3 Used in This Study. 2026. Available online: https://github.com/yandex-research/tabm (accessed on 31 July 2026).
43. Collins, G.S.; Dhiman, P.; Ma, J.; Schlussel, M.M.; Archer, L.; Van Calster, B.; Harrell, F.E.; Martin, G.P.; Moons, K.G.M.; van Smeden, M.; et al. Evaluation of Clinical Prediction Models (Part 1): From Development to External Validation. *BMJ* **2024**, *384*, e074819. https://doi.org/10.1136/bmj-2023-074819.
44. Tashman, L.J. Out-of-Sample Tests of Forecasting Accuracy: An Analysis and Review. *Int. J. Forecast.* **2000**, *16*, 437–450. https://doi.org/10.1016/S0169-2070(00)00065-0.
45. Gneiting, T.; Raftery, A.E. Strictly Proper Scoring Rules, Prediction, and Estimation. *J. Am. Stat. Assoc.* **2007**, *102*, 359–378. https://doi.org/10.1198/016214506000001437.
46. Yang, W.; Jiang, J.; Schnellinger, E.M.; Kimmel, S.E.; Guo, W. Modified Brier Score for Evaluating Prediction Accuracy for Binary Outcomes. *Stat. Methods Med Res.* **2022**, *31*, 2287–2296. https://doi.org/10.1177/09622802221122391.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.