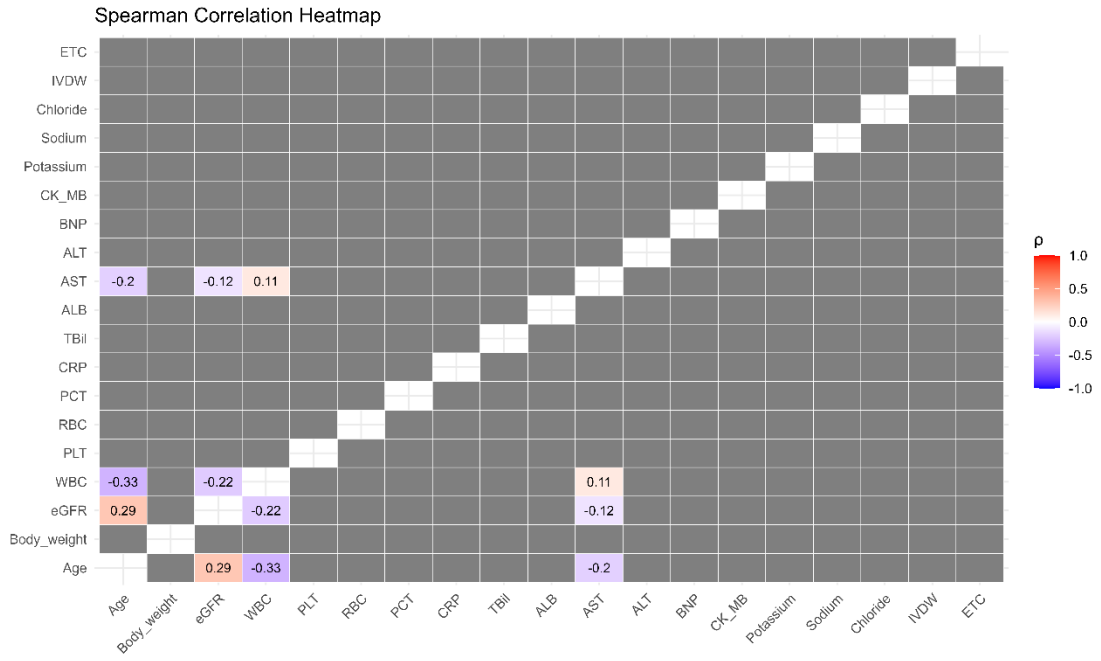
Supplementary Materials

Supplementary Figures

Supplementary Tables

Supplementary codes

Figure S1. Spearman correlation heatmap of features



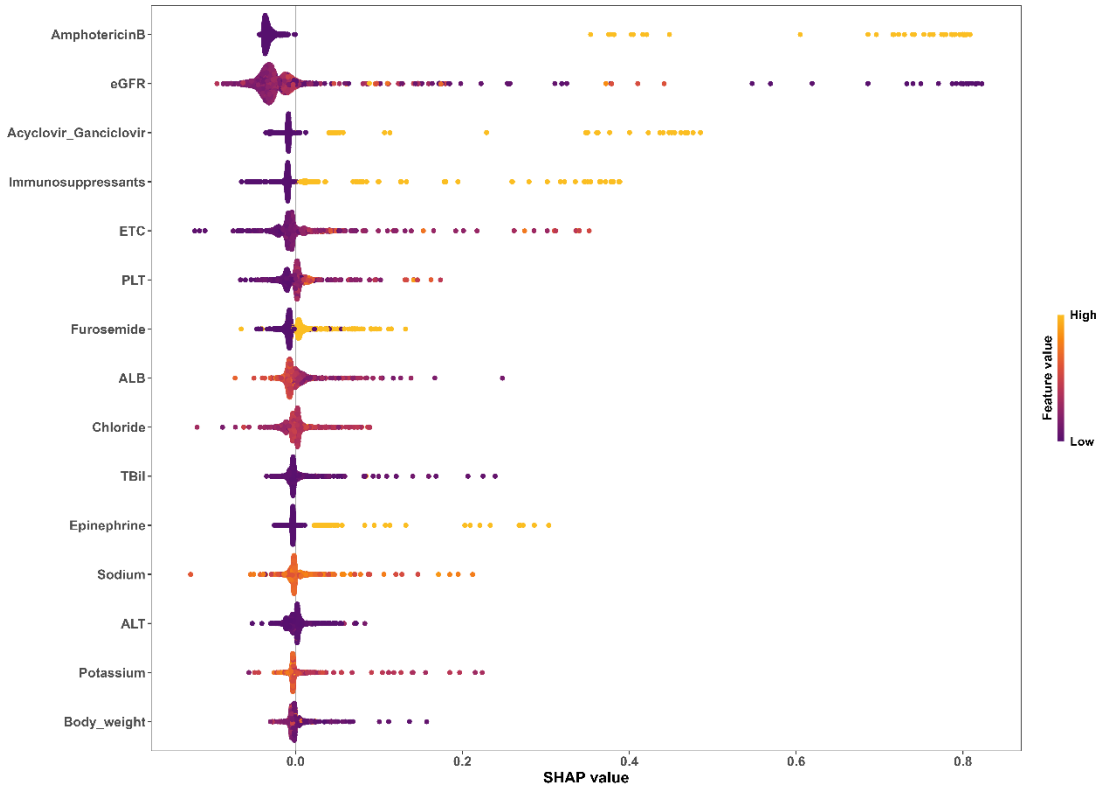
The heatmap displays pairwise Spearman rank correlation coefficients (ρ) among 19 variables, including age, body weight, estimated glomerular filtration rate (eGFR), white blood cell count (WBC), platelet count (PLT), red blood cell count (RBC), procalcitonin (PCT), C-reactive protein (CRP), total bilirubin (TBil), albumin (ALB), alanine aminotransferase (ALT), aspartate aminotransferase (AST), B-type natriuretic peptide (BNP), creatine kinase-MB (CK-MB), potassium, sodium, chloride, initial vancomycin dosage per weight (IVDW), and early vancomycin trough concentration (ETC). Only the lower triangular matrix is presented; the diagonal is left blank. The color scale ranges from blue (ρ = -1.0) through white (ρ = 0) to red (ρ = 1.0), with deeper color intensity indicating stronger correlation. Notable correlations include weak positive correlations between age and eGFR

(ρ = 0.29), and weak negative correlations between age and WBC (ρ = -0.33), age and AST (ρ = -0.20), and eGFR and WBC (ρ = -0.22).

Figure S2. DeLong's test P-value matrix for pairwise comparison of machine learning models

The heatmap displays pairwise P-values from DeLong's tests comparing the area under the receiver operating characteristic curve (AUROC) among eight machine learning models: decision tree (DT), K-nearest neighbors (KNN), neural network (NNET), random forest (RF), AdaBoost, LightGBM, support vector machine (SVM), and XGBoost. Only the lower triangular matrix is shown; the diagonal is left blank. The color scale ranges from yellow (P = 0.00) to green (P = 1.00), with greener colors indicating larger P-values (less significant differences). Significant differences ($P < 0.05$) are highlighted in yellow.

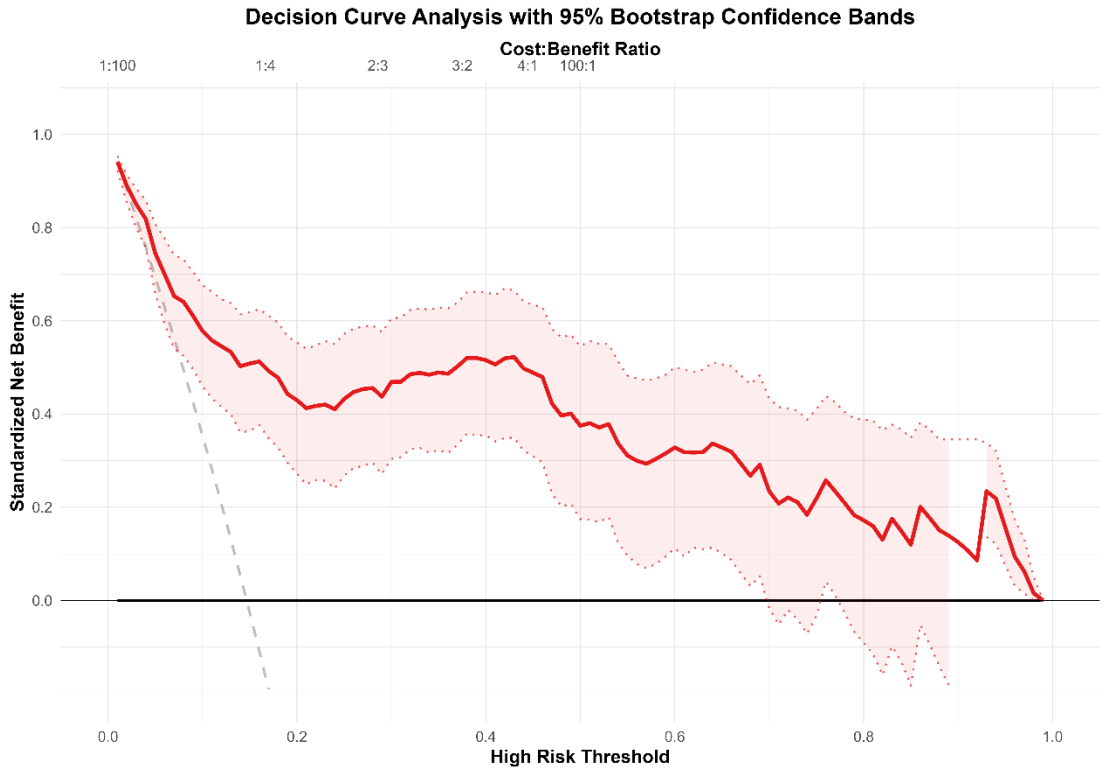
Figure S3. SHAP summary plot illustrating feature contributions to the XGBoost model for predicting vancomycin-induced nephrotoxicity.



The beeswarm plot displays the top 15 ranked features based on their mean absolute SHAP values from the machine learning model. Each dot represents an individual sample, with its horizontal position indicating the SHAP value (contribution to the model output) and its color representing the corresponding feature value (purple = low, yellow = high). Features are ranked from top to bottom by their overall importance. Amphotericin B, estimated glomerular filtration rate (eGFR), acyclovir/ganciclovir, immunosuppressants, and early vancomycin trough concentration (ETC) were identified as the five

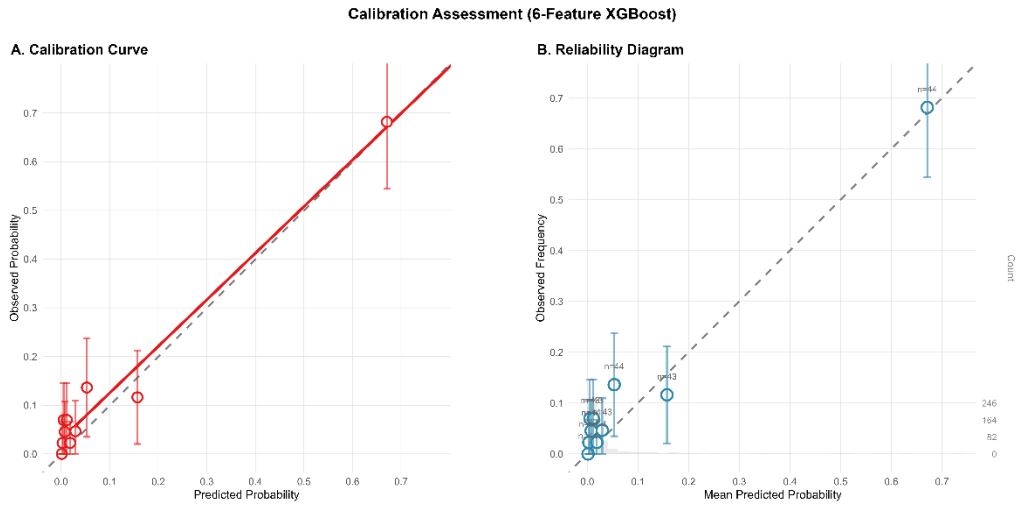
most influential predictors.

Figure S4 Decision curve analysis of the final tuned XGBoost model for predicting VIN.



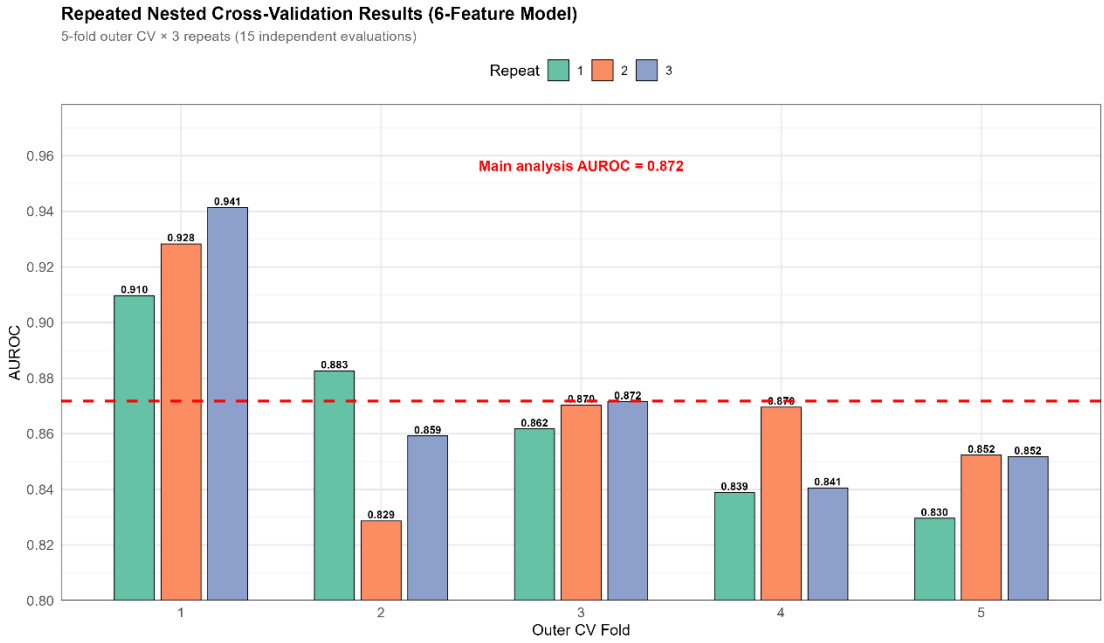
Standardized net benefit of the 6-feature XGBoost model (red line) across risk thresholds, with 95% confidence intervals from 1,000 bootstrap resamples (red dotted lines and shaded area). Gray dashed line: treat all; black line: treat none. Upper x-axis: high-risk threshold; lower x-axis: cost:benefit ratio.

Figure S5 Calibration assessment of the 6-feature XGBoost model.



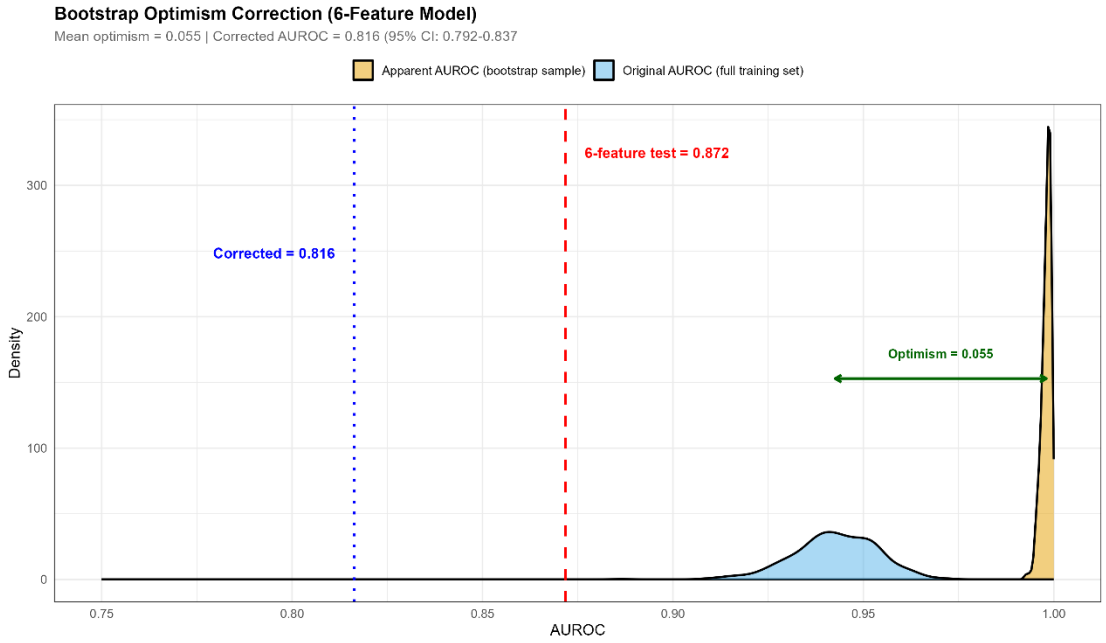
(A) Calibration curve showing the relationship between predicted probability and observed probability. The solid red line represents the fitted calibration curve, and the dashed gray line indicates the ideal calibration (perfect agreement). Error bars represent 95% confidence intervals. The model demonstrated good overall calibration, with the calibration curve closely following the ideal line across most of the probability range. (B) Reliability diagram displaying the observed frequency versus mean predicted probability within each bin. The size of each circle is proportional to the number of samples in that bin (n indicated), and error bars represent 95% confidence intervals. The dashed gray line denotes perfect calibration.

Figure S6. Repeated Nested Cross-validation Results.



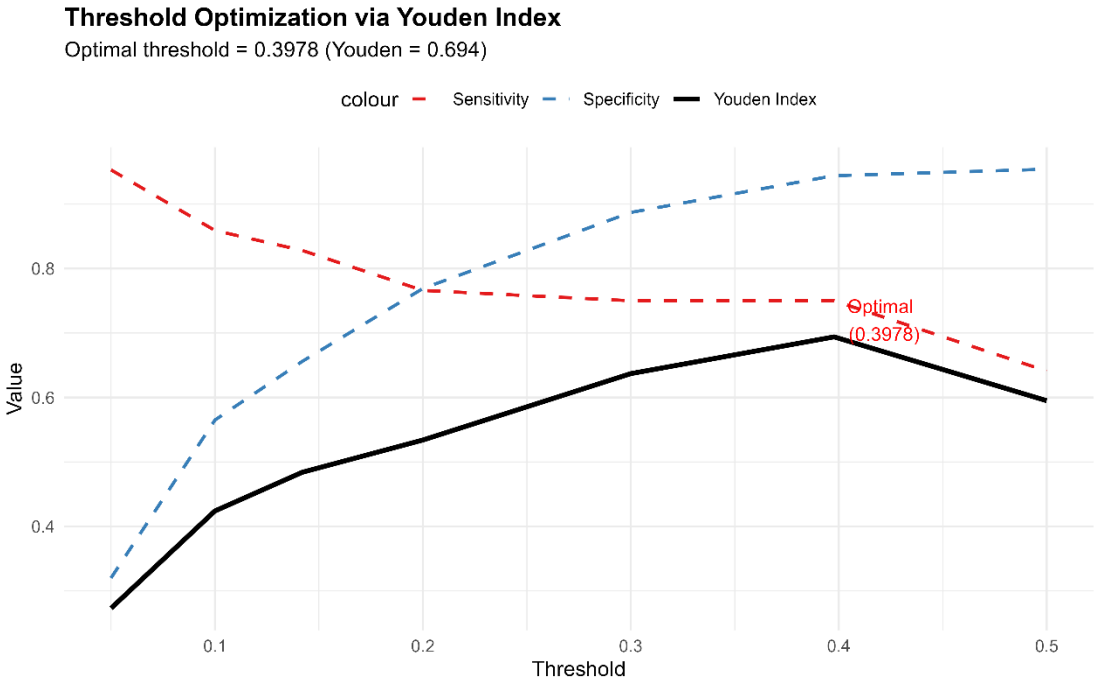
A 5-fold outer cross-validation procedure was repeated 3 times to generate 15 independent assessments. Colored bars show fold-level AUROC for each replicate run (Repeat 1, green; Repeat 2, orange; Repeat 3, blue). The red dashed line denotes the overall pooled mean AUROC of 0.872 from the main analysis, with individual AUROC values annotated on each bar.

Figure S7. Bootstrap-based optimism correction for AUROC of the 6-feature predictive model



Kernel density distributions of apparent AUROC (yellow, obtained from bootstrap resampling datasets) and original training-set AUROC (light blue, derived from the full training cohort) are displayed on the horizontal axis (AUROC) with density on the vertical axis. The red dashed vertical line marks the mean AUROC (0.872). The blue dotted vertical line indicates the optimism-corrected AUROC (0.816). The horizontal green arrow quantifies the mean optimism value of 0.055, representing the upward bias of model performance observed in the original training data. Global summary metrics are annotated at the top: mean optimism = 0.055, corrected AUROC = 0.816 (95% CI: 0.792–0.837).

Figure S8. Threshold optimization via Youden index.



The Youden index (J = sensitivity + specificity − 1, black solid line) was calculated across the clinically evaluated threshold range (0.05–0.50) to identify the optimal operating point. Sensitivity (red dashed line) and specificity (blue dashed line) curves demonstrate the trade-off between true positive and true negative rates. The optimal threshold (0.3978, J = 0.694) is marked by the red circle, where the Youden index reaches its maximum. This threshold balances diagnostic discrimination with clinical feasibility, yielding sensitivity 75.0% and specificity 94.4% on the independent test set.

Table S1. Optimal hyperparameters for all models

Models	hyperparameter search ranges	Optimal hyperparameters
DT	cp: 0.01 – 0.1(logscale = TRUE); maxdepth: 4 – 20; minsplit: 10 – 30	cp: -4.61; maxdepth: 12; minsplit: 20
KNN	k: 2 – 50; distance: 1 – 3; kernel: rectangular, triangular, epanechnikov;	k: 26; distance: 1; kernel: triangular
NNET	size: 1 – 20; decay: 0.0001 – 0.1(logscale = TRUE); maxit: 100 – 1000	decay: -2.30; maxit: 100; size: 20
RF	num.trees: 50 – 500; mtry: 1 – 7; min.node.size: 1 – 10; splitrule: gini, extratrees; sample.fraction: 0.1 – 1.0	min.node.size: 1; mtry: 4; num.trees: 500; sample. fraction: 1; splitrule: extratrees
Adaboost	I: 10 – 200; S: 1 – 5	S:1; I: 200
LightGBM	n.trees: 100 – 500; interaction.depth: 1 – 10; shrinkage: 0.001 – 0.2; bag.fraction: 0.5 – 1.0; n.minobsinnode: 5 – 50	n.trees: 300; interaction.depth: 10; n.minobsinnode:5; shrinkage: 0.2; bag.fraction: 0.5
SVM	cost: 1e-5 – 1e5(logscale = TRUE); gamma: 1e-5 – 1e5 (logscale = TRUE)	cost: 11.51; gama:0
XGBoost	nrounds: 50 – 500; eta: 0.01 – 0.3 (logscale = TRUE); max_depth: 3 – 10; min_child_weight: 1 – 10; subsample: 0.5 – 1.0; colsample_bytree: 0.5 – 1.0; lambda: 0 – 10; alpha: 0 – 10; gamma: 0 – 10	alpha:0; colsample_bytree: 1; eta: -2.90; gama:0; lambda: 0; max_depth: 10; min_chidl_weight: 1; nrouds: 500; subsample: 0.5

Table S2. The TRIPOD-AI checklist

Section/Topic	Item	Development / evaluation	Checklist item	Reported on page
TITLE				
Title	1	D;E	Identify the study as developing or evaluating the performance of a multivariable prediction model, the target population, and the outcome to be predicted	Title Page— "Development and Internal Validation of an Interpretable Machine Learning Model for Predicting Vancomycin-Induced Nephrotoxicity in Hospitalized Children"
ABSTRACT				
Abstract	2	D;E	See TRIPOD+AI for Abstracts checklist	page 2-4
INTRODUCTION				
Background	3a	D;E	Explain the healthcare context (including whether diagnostic or prognostic) and rationale for developing or evaluating the prediction model, including references to existing models	page 4-6
	3b	D;E	Describe the target population and the intended purpose of the prediction model in the context of the care pathway, including its intended users (e.g., healthcare professionals, patients, public)	page 6-7; intended purpose: "decision-support framework for identifying high-risk patients requiring enhanced therapeutic drug monitoring"; intended users: clinicians (pharmacists, physicians)

	3c	D;E	Describe any known health inequalities between sociodemographic groups	-
Objectives	4	D;E	Specify the study objectives, including whether the study describes the development or validation of a prediction model (or both)	page 6
METHODS				page 6-7
Data	5a	D;E	Describe the sources of data separately for the development and evaluation datasets (e.g., randomised trial, cohort, routine care or registry data), the rationale for using these data, and representativeness of the data	Data source: retrospective EHR-based study at Children's Hospital of Chongqing Medical University (tertiary children's hospital); Rationale: large pediatric cohort with systematic vancomycin TDM; Representativeness: single-center, may limit generalizability to other settings; Development dataset: training set (70%, n=1,016); Evaluation dataset: test set (30%, n=436)
	5b	D;E	Specify the dates of the collected participant data, including start and end of participant accrual; and, if applicable, end of follow-up	page 6-7 ".…….between December 2017 and November 2024"; no end of follow-up specified as outcome assessed during vancomycin treatment
Participants	6a	D;E	Specify key elements of the study setting (e.g., primary care, secondary care, general population) including the number and location of centres	page 6-7 Setting: tertiary children's hospital; Number and location of centers: 1 center (Children's Hospital of

				Chongqing Medical University, Chongqing, China) page 6-7
	6b	D;E	Describe the eligibility criteria for study participants	Detailed inclusion criteria: age 1 month to 18 years, vancomycin ≥3 consecutive days, baseline renal function test; Exclusion criteria: major missing data, eGFR ≤59 mL/min/1.73m^2, preterm infants, blood purification therapy, lack of baseline eGFR page 7-8
	6c	D;E	Give details of any treatments received, and how they were handled during model development or evaluation, if relevant	Vancomycin dosing (IVDW), concomitant medications (amphotericin B, immunosuppressants, etc.) handled as predictor variables; no treatment effect estimation performed page 7-8
Data preparation	7	D;E	Describe any data pre-processing and quality checking, including whether this was similar across relevant sociodemographic groups	Pre-processing: missing value exclusion (>25%), multicollinearity removal (Spearman ρ>0.6), standardization, treatment encoding; Quality checking: temporal validation of predictor-outcome sequence
Outcome	8a	D;E	Clearly define the outcome that is being predicted and the time horizon, including how and when assessed, the	page 7 Outcome: VIN defined as Scr increase ≥0.5 mg/dL or ≥50% from baseline on

			rationale for choosing this outcome, and whether the method of outcome assessment is consistent across sociodemographic groups	≥2 consecutive measurements during treatment; Time horizon: during vancomycin treatment
	8b	D;E	If outcome assessment requires subjective interpretation, describe the qualifications and demographic characteristics of the outcome assessors	Not applicable Outcome based on objective laboratory measurements (serum creatinine); no subjective interpretation required
	8c	D;E	Report any actions to blind assessment of the outcome to be predicted	Not reported Retrospective study using EHR data; no blinding procedures described; outcome assessment based on laboratory values unlikely to be influenced by predictor knowledge
Predictors	9a	D	Describe the choice of initial predictors (e.g., literature, previous models, all available predictors) and any pre-selection of predictors before model building	page 7-8 Initial predictors: 34 features selected based on literature review, clinical relevance, and EHR availability; Pre-selection: (1) exclusion of features with >25% missing values; (2) Spearman correlation analysis (ρ>0.6) retaining feature more strongly associated with outcome
	9b	D;E	Clearly define all predictors, including how and when they were measured	page 7-8

			(and any actions to blind assessment of predictors for the outcome and other predictors)	All 34 predictors defined: baseline demographics and laboratory variables (within 24h before initiation), ETC (first trough within 48h), IVDW, concurrent medications; **Blinding of predictor assessment not reported**
	9c	D;E	If predictor measurement requires subjective interpretation, describe the qualifications and demographic characteristics of the predictor assessors	Not applicable All predictors are objective (laboratory values, medication records, demographic data); no subjective interpretation required
Sample size	10	D;E	Explain how the study size was arrived at (separately for development and evaluation), and justify that the study size was sufficient to answer the research question. Include details of any sample size calculation	-
Missing data	11	D;E	Describe how missing data were handled. Provide reasons for omitting any data	page 7-8 Missing data handling: (1) exclusion of features with >25% missing values (justification: established guidance that excessive missingness introduces bias); (2) histogram-based imputation for numeric variables; (3) mode imputation for categorical variables; Reasons for omitting participant data:

				missing height (1,071 patients, 32.1%), preterm birth, eGFR<59, etc. detailed in Results page 8-9
Analytical methods	12a	D	Describe how the data were used (e.g., for development and evaluation of model performance) in the analysis, including whether the data were partitioned, considering any sample size requirements	Data partitioning: 70/30 split with fixed random seed (set.seed=123); Training set used for: preprocessing, SMOTE, hyperparameter tuning (3-fold CV); Test set: locked, evaluated once; **Sample size considerations for partitioning not explicitly discussed** page 7-8
	12b	D	Depending on the type of model, describe how predictors were handled in the analyses (functional form, rescaling, transformation, or any standardisation)	Numeric variables: standardization; Categorical variables: treatment encoding; Missing values: histogram imputation (numeric), mode imputation (categorical); Class imbalance: SMOTE (dup_size=2) page 8-9
	12c	D	Specify the type of model, rationale, all model-building steps, including any hyperparameter tuning, and method for internal validation	Model types: 8 ML algorithms (DT, KNN, RF, AdaBoost, LightGBM, NNET, SVM, XGBoost); Rationale: systematic comparison to identify optimal approach; Model-building: 3-fold CV with grid search + manual fine-tuning; page 9-10

			Internal validation: (1) repeated nested CV (5-fold outer × 3 repeats, stratified); (2) bootstrap optimism correction (500 resamples)
12d	D;E	Describe if and how any heterogeneity in estimates of model parameter values and model performance was handled and quantified across clusters (e.g., hospitals, countries). See TRIPOD-Cluster for additional considerations	-
12e	D;E	Specify all measures and plots used (and their rationale) to evaluate model performance (e.g., discrimination, calibration, clinical utility) and, if relevant, to compare multiple models	page 9-10 Discrimination: AUROC (DeLong's test for pairwise comparison); Classification: sensitivity, specificity, PPV, NPV, ACC, F1; Precision-recall: AUPR; Calibration: Brier score; Clinical utility: DCA; Interpretability: SHAP (beeswarm, waterfall plots), gain-based feature importance (100 bootstrap replicates); Calibration assessment: reliability diagram, calibration slope and intercept
12f	E	Describe any model updating (e.g., recalibration) arising from the model evaluation, either overall or for	-

			particular sociodemographic groups or settings	
	12g	E	For model evaluation, describe how the model predictions were calculated (e.g., formula, code, object, application programming interface)	page 9 "The complete analytical code is provided in Supplementary Materials (Microsoft Word format)"; mlr3 ecosystem in R version 4.4.2;
Class imbalance	13	D;E	If class imbalance methods were used, state why and how this was done, and any subsequent methods to recalibrate the model or the model predictions	page 8-9 SMOTE used (dup_size=2); Why: VIN rate 14.2%, class imbalance; How: synthetic minority oversampling to achieve approximately 1:1 ratio;
Fairness	14	D;E	Describe any approaches that were used to address model fairness and their rationale	-
Model output	15	D	Specify the output of the prediction model (e.g., probabilities, classification). Provide details and rationale for any classification and how the thresholds were identified	page 10-11 Output: predicted probabilities; Classification: binary (VIN+ / VIN−); Threshold selection: (1) baseline event rate (14.2%); (2) Youden index (J = sensitivity + specificity − 1), optimal threshold 0.3978; (3) DCA; Rationale: "clinically relevant range (0.05–0.50)"
Training versus evaluation	16	D;E	Identify any differences between the development and evaluation data in	Random split from same source population; Table S3 shows no significant differences in baseline

			healthcare setting, eligibility criteria, outcome, and predictors	characteristics between training and test sets. page 11
Ethical approval	17	D;E	Name the institutional research board or ethics committee that approved the study and describe the participant-informed consent or the ethics committee waiver of informed consent	"Ethics Committee of the Children's Hospital of Chongqing Medical University (No. 202513; approved on January 15, 2025)"; Waiver of informed consent granted due to retrospective design and anonymized data
OPEN SCIENCE				
Funding	18a	D;E	Give the source of funding and the role of the funders for the present study	Title Page / Declarations "Pharmacovigilance Open Project of Sichuan Center for Food and Drug Evaluation and Safety Monitoring (Grant No. YWJJ202508)"; **Role of funders not reported**
Conflicts of interest	18b	D;E	Declare any conflicts of interest and financial disclosures for all authors	Title Page / Declarations "All authors declared no competing interests for this work"
Protocol	18c	D;E	Indicate where the study protocol can be accessed or state that a protocol was not prepared	A protocol was not prepared for this retrospective study
Registration	18d	D;E	Provide registration information for the study, including register name and registration number, or state that the study was not registered	This study was not registered

Data sharing	18e	D;E	Provide details of the availability of the study data	Declarations— "available from the corresponding author upon reasonable request"
Code sharing	18f	D;E	Provide details of the availability of the analytical code	complete analytical code is provided in Supplementary Materials (Microsoft Word format)
Patient & Public Involvement				
Patient & Public Involvement	19	D;E	Provide details of any patient and public involvement during the design, conduct, reporting, interpretation, or dissemination of the study or state no involvement	No patient or public involvement
RESULTS				
Participants	20a	D;E	Describe the flow of participants through the study, including the number of participants with and without the outcome and, if applicable, a summary of the follow-up time. A diagram may be helpful	Page 13-14 Figure 1 (Study flow diagram): 3,338 initially identified → 1,886 excluded (detailed reasons) → 1,452 included (206 VIN, 1,246 non-VIN) → 1,016 training / 436 test; No follow-up time as outcome assessed during treatment
	20b	D;E	Report the characteristics overall and, where applicable, for each data source or setting, including the key dates, key predictors (including demographics), treatments received, sample size, number of outcome events, follow-up	Table 1 Overall characteristics: VIN vs non-VIN groups (median, IQR, n, %); Table S2 Training vs test set characteristics; Key dates: December 2017–November

			time, and amount of missing data. A table may be helpful. Report any differences across key demographic groups	2024; Missing data: reported for each variable in Table 1
	20c	E	For model evaluation, show a comparison with the development data of the distribution of important predictors (demographics, predictors, and outcome)	Supplementary Material Table S2 Comparison of training and test set distributions;
Model development	21	D;E	Specify the number of participants and outcome events in each analysis (e.g., for model development, hyperparameter tuning, model evaluation)	Page 14-15 Development: 1,016 participants (142 VIN events); Hyperparameter tuning: 3-fold CV within training set; Evaluation: 436 participants (64 VIN events); SMOTE-balanced training: ~2,000 synthetic samples
Model specification	22	D	Provide details of the full prediction model (e.g., formula, code, object, application programming interface) to allow predictions in new individuals and to enable third-party evaluation and implementation, including any restrictions to access or re-use	Supplementary Materials R code provided in Word format; No prediction formula, model object, or API provided
Model performance	23a	D;E	Report model performance estimates with confidence intervals, including for any key subgroups (e.g.,	Page 19 AUROC 0.87 (95% CI: 0.816–0.927); Sensitivity 75.0% (95% CI: 62.8%–

			sociodemographic). Consider plots to aid presentation	84.5%); Specificity 94.4% (95% CI: 91.7%–96.4%); PPV 69.6% (95% CI: 57.3%–79.9%); NPV 95.6% (95% CI: 93.1%–97.3%); Subgroup performance not reported; no stratification by age, sex, or disease severity
	23b	D;E	If examined, report results of any heterogeneity in model performance across clusters. See TRIPOD Cluster for additional details	Single-center study; no cluster heterogeneity examined
Model updating	24	E	Report the results from any model updating, including the updated model and subsequent performance	Not applicable — No model updating performed
DISCUSSION				
Interpretation	25	D;E	Give an overall interpretation of the main results, including issues of fairness in the context of the objectives and previous studies	Page 20 Overall interpretation provided; comparison with prior studies (AUROC 0.66–0.83); **Fairness issues not discussed**
Limitations	26	D;E	Discuss any limitations of the study (such as a non-representative sample, sample size, overfitting, missing data) and their effects on any biases, statistical uncertainty, and generalizability	Page 22-23 Eight limitations discussed: (1) single-center design and selection bias; (2) unmeasured confounding by disease severity; (3) moderate sensitivity; (4) SHAP predictive not causal; (5) creatinine-based outcome limitations;

				(6) missing data exclusion; (7) no temporal validation (calibration drift, temporal drift); (8) calibration slope under-confidence
	27a	D	Describe how poor quality or unavailable input data (e.g., predictor values) should be assessed and handled when implementing the prediction model	-
	27b	D	Specify whether users will be required to interact in the handling of the input data or use of the model, and what level of expertise is required of users	-
	27c	D;E	Discuss any next steps for future research, with a specific view to applicability and generalizability of the model	Page 27-28 Six directions: (1) prospective multicenter validation; (2) novel renal biomarkers (cystatin C); (3) real-time EHR deployment; (4) comparative effectiveness research; (5) temporal model updating; (6) extension to neonatal populations

Table S3. Baseline characteristics of included patients versus patients with missing eGFR data.

variables	eGFR=yes (n=1452)	eGFR=no (n=1071)	P value
ETC	6.97 (3.97 – 9.97)	6.95 (3.96 – 9.94)	0.958
Amphotericin B	66 (4.5%)	49 (4.6%)	0.912
Immunosuppressants	192 (13.2%)	142 (13.3%)	0.945
Acyclovir/Ganciclovir	75 (5.2%)	55 (5.1%)	0.928
ALB	36.40 (31.65 – 41.15)	35.90 (31.45 – 40.35)	0.530
Furosemide	582 (40.1%)	345 (32.2%)	**<0.001**
Epinephrine	54 (3.7%)	30 (2.8%)	0.247
Norepinephrine	97 (6.7%)	57 (5.3%)	0.185
Piperacillin/tazobactam	99 (6.8%)	115 (10.7%)	**<0.001**
NSAIDs	897 (61.8%)	748 (69.8%)	**<0.001**
Aminoglycosides	9 (0.6%)	2 (0.2%)	0.185
Chemotherapy	140 (9.6%)	55 (5.1%)	**<0.001**
Acetaminophen	239 (16.5%)	120 (11.2%)	**<0.001**
ACEI	47 (3.2%)	70 (6.5%)	**<0.001**
Other beta-lactam	1245 (85.7%)	871 (81.3%)	**0.003**
Quinolones	71 (4.9%)	15 (1.4%)	**<0.001**
SMZ/TMP	73 (5.0%)	9 (0.8%)	**<0.001**
Bod weight	13.00 (5.25 – 20.75)	17.50 (9.75 – 25.25)	**<0.001**
WBC	7.84 (1.49 – 14.20)	9.79 (4.73 – 14.86)	**<0.001**
RBC	3.41 (2.88 – 3.95)	3.64 (3.14 – 4.15)	**<0.001**
PLT	215.00 (41.50 – 388.50)	258.00 (116.50 – 399.50)	**<0.001**
PCT	0.44 (0.00 – 2.34)	0.46 (0.00 – 2.32)	0.236
CRP	29.70 (1.50 – 57.90)	32.00 (5.66 – 58.34)	0.289
TBil	8.60 (3.87 – 13.34)	7.10 (3.50 – 10.70)	**<0.001**
AST	32.00 (16.00 – 48.00)	29.60 (17.40 – 41.80)	**0.004**
ALT	27.00 (10.28 – 43.73)	25.50 (12.50 – 38.50)	0.153
CK-MB	0.71 (0.00 – 1.54)	0.73 (0.00 – 1.46)	0.949
Potassium	4.12 (3.72 – 4.53)	4.12 (3.77 – 4.47)	0.262
Sodium	137.00 (134.63 – 139.38)	137.00 (134.82 – 139.19)	0.195
Chloride	102.60 (100.10 – 105.10)	102.00 (99.29 – 104.71)	**0.009**

IVDW	10.26 (8.44 – 12.08)	10.24 (8.43 – 12.05)	0.962

Data are presented as median (interquartile range), or number (percentage), as appropriate. Continuous variables were compared using the t-test or Mann–Whitney U test, while categorical variables were compared using the χ^2 test or Fisher's exact test, as applicable. P-values < 0.05 were considered statistically significant. Abbreviations: eGFR, estimated glomerular filtration rate; WBC, white blood cell count; PLT, platelet count; RBC, red blood cell count; ALB, albumin; PCT, procalcitonin; TBil, total bilirubin; ALT, alanine transaminase; AST, aspartate aminotransferase; NSAIDs, non-steroidal anti-inflammatory drugs; ACEI, angiotensin-converting enzyme inhibitor; SMZ/TMP, sulfamethoxazole-trimethoprim; ETC, early vancomycin trough concentration; IVDW, initial vancomycin dosage per weight ; CK-MB, creatine kinase-myocardial band. Bold numbers indicate statistically significant differences.

Table S4. Baseline characteristics of patients in the training and test sets.

Variable	N	test N = 436	train N = 1,016	p-value
Nephrotoxicity, n (%)	1,452			0.73
yes		64 (15)	142 (14)	
no		372 (85)	874 (86)	
ACEI, n (%)	1,452			0.18
no		426 (98)	979 (96)	
yes		10 (2.3)	37 (3.6)	
ALB, Median (IQR)	1,449	36.30 (30.30 – 40.60)	36.40 (31.20 – 40.60)	0.45
ALT, Median (IQR)	1,451	26.00 (16.30 – 48.90)	27.80 (16.70 – 50.40)	0.32
AST, Median (IQR)	1,452	32.00 (23.00 – 56.35)	32.00 (22.00 – 52.00)	0.60
Acetaminophen, n (%)	1,452			**0.049**
no		377 (86)	836 (82)	
yes		59 (14)	180 (18)	
Acyclovir/Ganciclovir, n (%)	1,452			0.52
no		411 (94)	966 (95)	
yes		25 (5.7)	50 (4.9)	
Age, Median (IQR)	1,452	2.35 (0.69 – 7.44)	2.87 (0.78 – 8.05)	0.31

Variable	N	test N = 436	train N = 1,016	p-value
Aminoglycosides, n (%)	1,452			>0.99
no		433 (99)	1,010 (99)	
yes		3 (0.7)	6 (0.6)	
Amphotericin B, n (%)	1,452			0.75
no		415 (95)	971 (96)	
yes		21 (4.8)	45 (4.4)	
Body weight, Median (IQR)	1,450	12.00 (8.00 – 21.00)	13.60 (8.00 – 25.00)	0.21
CK-MB, Median (IQR)	1,201	0.75 (0.18 – 1.63)	0.70 (0.18 – 1.75)	0.98
CRP, Median (IQR)	1,443	33.00 (11.00 – 71.25)	28.00 (8.00 – 62.00)	**0.016**
Chemotherapy, n (%)	1,452			0.43
no		398 (91)	914 (90)	
yes		38 (8.7)	102 (10)	
Chloride, Median (IQR)	1,421	102.30 (99.60 – 104.80)	102.70 (100.20 – 105.10)	0.11
ETC, Median (IQR)	1,170	6.86 (4.68 – 10.60)	7.03 (4.39 – 10.36)	0.86
Epinephrine, n (%)	1,452			0.59
no		418 (96)	980 (96)	

Variable	N	test N = 436	train N = 1,016	p-value
yes		18 (4.1)	36 (3.5)	
Furosemide, n (%)	1,452			0.84
no		263 (60)	607 (60)	
yes		173 (40)	409 (40)	
IVDW, Median (IQR)	1,450	10.24 (10.00 – 13.85)	10.26 (10.00 – 13.51)	0.99
Immunosuppressants, n (%)	1,452			0.26
no		385 (88)	875 (86)	
yes		51 (12)	141 (14)	
NSAIDs, n (%)	1,452			0.31
no		158 (36)	397 (39)	
yes		278 (64)	619 (61)	
Norepinephrine, n (%)	1,452			0.84
no		406 (93)	949 (93)	
yes		30 (6.9)	67 (6.6)	
Other beta-lactam, n (%)	1,452			0.98
no		62 (14)	145 (14)	

Variable	N	test N = 436	train N = 1,016	p-value
yes		374 (86)	871 (86)	
PCT, Median (IQR)	1,361	0.58 (0.17 – 3.18)	0.39 (0.15 – 2.16)	0.093
PLT, Median (IQR)	1,451	228.00 (81.50 – 403.50)	203.00 (48.00 – 402.00)	0.19
Piperacillin/tazobactam, n (%)	1,452			0.23
no		401 (92)	952 (94)	
yes		35 (8.0)	64 (6.3)	
Potassium, Median (IQR)	1,438	4.13 (3.78 – 4.53)	4.11 (3.73 – 4.57)	0.87
Quinolones, n (%)	1,452			0.73
no		416 (95)	965 (95)	
yes		20 (4.6)	51 (5.0)	
RBC, Median (IQR)	1,445	3.47 (2.96 – 4.02)	3.38 (2.85 – 3.94)	0.075
SMZ/TMP, n (%)	1,452			0.20
no		419 (96)	960 (94)	
yes		17 (3.9)	56 (5.5)	
Sex, n (%)	1,452			0.28
Female		164 (38)	413 (41)	

Variable	N	test N = 436	train N = 1,016	p-value
Male		272 (62)	603 (59)	
Sodium, Median (IQR)	1,439	136.80 (134.00 – 139.00)	137.00 (134.90 – 139.00)	0.12
TBil, Median (IQR)	1,450	8.50 (5.40 – 14.65)	8.60 (5.00 – 14.30)	0.59
WBC, Median (IQR)	1,452	8.38 (1.31 – 13.79)	7.65 (0.91 – 13.64)	0.12
eGFR, Median (IQR)	1,452	131.43 (106.89 – 162.26)	130.59 (109.16 – 157.36)	0.72

Continuous variables are presented as median (interquartile range) and compared using the t-test or Mann-Whitney U test. Categorical variables are presented as n (%) and compared using the Chi-squared or Fisher's exact test, as appropriate. P-values < 0.05 were considered statistically significant. Abbreviations: eGFR, estimated glomerular filtration rate; WBC, white blood cell count; PLT, platelet count; RBC, red blood cell count; ALB, albumin; PCT, procalcitonin; TBil, total bilirubin; ALT, alanine transaminase; AST, aspartate aminotransferase; NSAIDs, non-steroidal anti-inflammatory drugs; ACEI, angiotensin-converting enzyme inhibitor; SMZ/TMP, sulfamethoxazole-trimethoprim; ETC, early vancomycin trough concentration; IVDW, Initial vancomycin dosage per weight; CK-MB, creatine kinase-myocardial ban; CRP, C-reactive protein. Bold numbers indicate statistically significant differences.

Table S5. Delong test for pairwise AUC comparison of 6-/8-/10-/full-feature models in XGBoost model

Comparison	AUC Differences	Z value	P value	Significance
Full features vs 10 features	0.023	0.91	0.36	ns
Full features vs 8 features	0.036	1.64	0.10	ns
Full features vs 6 features	0.025	1.07	0.29	ns
10 features vs 8 features	0.013	0.53	0.60	ns
10 features vs 6 features	0.0016	0.07	0.95	ns
8 features vs 6 features	-0.012	-0.55	0.58	ns

ns: non significance

Table S6. Detailed performance of 6-/8-/10-/full-feature models in XGBoost model

Feature number	AUROC	ACC	AUPR	Brier score	sensitivity	ppv	npv	F1
6	0.87	0.91	0.72	0.08	0.64	0.71	0.94	0.67
8	0.86	0.90	0.69	0.08	0.63	0.66	0.94	0.64
10	0.87	0.92	0.75	0.07	0.64	0.79	0.94	0.70
full	0.90	0.91	0.74	0.07	0.58	0.77	0.93	0.66

AUROC, area under the receiver operating characteristic curve; AUPR, area under the precision-recall curve; PPV, positive predictive value; NPV, negative predictive value.

Table S7. Predictive performance metrics of the 6-feature model from repeated nested cross-validation (3 repetitions × 5 outer folds, total 15 independent evaluations)

Repeat	AUROC_mean	AUROC_sd	AUROC_min	AUROC_max	Sensitivity_mean	Specificity_mean	Brier_mean
1	0.864523	0.032606	0.829619	0.909688	0.626365	0.94863	0.079639
2	0.869859	0.036799	0.828681	0.928299	0.650871	0.955839	0.07829
3	0.872909	0.039932	0.840533	0.941424	0.665389	0.947827	0.078526
15-folds	0.869097	0.034046	0.829009	0.936830	0.647542	0.950765	0.078818

Repeat 1/2/3 denote results from three separate runs of 5-fold outer cross-validation; the row labeled "15-folds" presents pooled metrics aggregated across all 15 outer validation folds. AUROC_mean: Mean area under the receiver operating characteristic curve across outer folds; AUROC_sd: Standard deviation of fold-level AUROC values; AUROC_min, AUROC_max: Minimum and maximum AUROC observed among all outer folds; Sensitivity_mean: Mean sensitivity (true positive rate) at the optimal Youden's index cut-off; Specificity_mean: Mean specificity (true negative rate) at the optimal Youden's index cut-off; Brier_mean: Mean Brier score, measuring calibration error (smaller values indicate better prediction calibration)

Table S8. Summary of bootstrap optimism correction metrics for the 6-feature predictive model

Metric	Value
Apparent AUROC (mean ± SD)	0.998 ± 0.001
Original Training AUROC (mean ± SD)	0.943 ± 0.011
Optimism AUROC (mean ± SD)	0.055 ± 0.012
Main Test AUROC (6-feature)	0.872
Optimism-Corrected AUROC	0.816
95% CI Lower	0.792
95% CI Upper	0.837

Apparent AUROC refers to the mean AUROC obtained from repeated bootstrap resampling datasets. Original Training AUROC denotes the model performance evaluated on the full original training cohort. Optimism AUROC represents the mean upward bias between apparent and training-set AUROC, quantifying overfitting-induced performance overestimation. Main Test AUROC corresponds to the pooled mean AUROC derived from repeated nested cross-validation (3 repetitions × 5 outer folds). Optimism-Corrected AUROC is the final bias-adjusted discrimination value after subtracting mean optimism, with its 95% confidence interval (CI) presented. SD = standard deviation.

Table S9. Confusion matrix at Youden-optimal threshold (0.3978)

	Predicted VIN+	Predicted VIN−	Total
Observed VIN+	48 (TP)	16 (FN)	64
Observed VIN−	21 (FP)	351 (TN)	372
Total	69	367	436

Supplementary Code

Development and Internal Validation of an Explainable Machine Learning Model
for Predicting Vancomycin-Induced Nephrotoxicity in Hospitalized Children

Software Requirements:
- R version 4.4.2
- Packages: mlr3, mlr3pipelines, mlr3tuning, mlr3learners, mlr3measures, openxlsx, dplyr

````
title: "Supplementary Code: Analytical Pipeline for VIN Prediction Model"
author: "Xiuling Wang et al."
date: "2026-06-17"
output:
  html_document:
    toc: true
    toc_float: true
    code_folding: show
    highlight: tango
    theme: flatly
  pdf_document:
    toc: true
    highlight: tango
  word_document:
    toc: true
    highlight: tango
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE, message = FALSE, warning = FALSE,
                      comment = NA, fig.align = "center")
```

# 1. Overview
This document provides the complete analytical code for the study:
**"Development and Internal Validation of an Explainable Machine Learning Model for Predicting Vancomycin-Induced Nephrotoxicity in Hospitalized Children"**
## Software Environment
- **R version**: 4.4.2
- **Key packages**: mlr3, mlr3pipelines, mlr3tuning, mlr3learners,
````

mlr3measures, openxlsx, DALEX, shap, pROC, rms, ggplot2

Step 1: Load software packages

```{r}
library(openxlsx)
library(tidyverse)
library(here)
library(mlr3verse)
library(tidyverse)
library(tidyfst)
library(DT)#datatable
library(mlr3viz)
library(DALEX)
library(DALEXtra)
library(kernelshap)
library(ggplot2)
library(permshap)
library(flextable)
library(rrtable)
library(autoReg)
library(smotefamily)
library(caret)
library(rms)
library(patchwork)
library(ggsci)
library(rmda)
library(shapviz)
library(pROC)
library(gtsummary)
library(data.table)
library(Cairo)
library(gridExtra)
library(grid)
```

Step 2: Baseline Table

import data

```{r}
df <- read.xlsx("Rawdata.xlsx") %>% select(-id,-BNP)
str(df)
set_gtsummary_theme(theme_gtsummary_journal("jama"))
```

```

```{r}
table_gtsummary <-
  df %>%
  tbl_summary(by = "Nephrotoxicity",
              missing = "no",
              type = list(where(is.factor) ~ "categorical"),
              digits = all_continuous() ~ 2,
  ) %>%
  add_n() %>%
  add_p(pvalue_fun = ~ style_pvalue(.x, digits = 2)) %>%
  modify_header(label = "**Variable**") %>%
  bold_labels() %>%
  separate_p_footnotes()
table_gtsummary
```

```{r}
table_gtsummary |>
  as_flex_table() |>
  flextable::save_as_docx(path = here("output"," Rework Baseline Table.docx"))
```

Baseline Data Comparison (Training Set & Test Set) ==========

import data

```{r}
df <- read.xlsx("Rawdata.xlsx") %>%
  select(-id,-BNP) %>%
  mutate(across(where(is.character), as.factor))
task <- as_task_classif(x=df,target = "Nephrotoxicity",id="test",positive = "yes")
task$positive
set.seed(123)
split=partition(task,ratio = 0.7)
```
```

Compare the training set with the test set

```{r}
task_train <- task$clone()$filter(rows = split$train)
task_test <- task$clone()$filter(rows = split$test)
data_train <-
   task_train$data() |>
   mutate(team="train") |>
   select(team,everything())
data_test <-
   task_test$data()|>
   mutate(team="test") |>
   select(team,everything())
```

```{r}
data_all <-
   bind_rows(data_train,data_test)

```

```{r}
table_gtsummary <-
   data_all %>%
   tbl_summary(by = "team",
                        missing = "no",
                        type = list(where(is.factor) ~ "categorical"),
                        digits = all_continuous() ~ 2,
   ) %>%
   add_n()
   add_p(pvalue_fun = ~ style_pvalue(.x, digits = 2)) %>%
   modify_header(label = "**Variable**") %>%
   bold_labels() %>%
   separate_p_footnotes()

table_gtsummary

```

```{r}
table_gtsummary |>
```

```
  as_flex_table() |>
  flextable::save_as_docx(path = here("output","
table_gtsummary_ML.docx"))
```

Step 3：Model Construction and Comparison

import data

```{r}
df <- read.xlsx("Rawdata.xlsx") %>%
  select(-id,-BNP) %>%
  mutate(across(where(is.character), as.factor))
df$Nephrotoxicity <- factor(ifelse(df$Nephrotoxicity == "yes", "1", "0"),
                            levels = c("0", "1"))

str(df)
```

```{r}
task <- as_task_classif(x=df,target =
"Nephrotoxicity",id="test",positive = "1")
task$positive
```
```{r}
global_pipeline <- po("imputehist") %>>%
  po("imputemode") %>>%
  po("scale", affect_columns = selector_type("numeric")) %>>%
  po("encode",method="treatment",affect_columns =
selector_type("factor"))
set.seed(123)
split <- partition(task, ratio = 0.7)
train_task <- task$clone()$filter(split$train)
test_task <- task$clone()$filter(split$test)
global_pipeline$train(train_task)
train_processed <- global_pipeline$predict(train_task)[[1]]
test_processed <- global_pipeline$predict(test_task)[[1]]
set.seed(123)
smote_pipe <- po("smote", dup_size = 2)
train_balanced <- smote_pipe$train(list(train_processed))[[1]]
measures = list(
  msr("classif.auc"),
```

```
  msr("classif.acc"),
  msr("classif.prauc"),
  msr("classif.bbrier"),
  msr("classif.sensitivity"),
  msr("classif.ppv"),
  msr("classif.npv"),
  msr("classif.fbeta")
)
# Create Decision Tree Learner
lrn_classif.rpart <- lrn("classif.rpart",predict_type = "prob")
lrn_classif.rpart$param_set
```

```{r}
set.seed(123)
```

```{r}
lrn_classif.rpart_2tune = lrn("classif.rpart",
                              cp = to_tune(p_dbl(lower = 0.01,upper = 0.1,logscale = T)),
                              maxdepth = to_tune(p_int(lower = 4,upper = 20)),
                              minsplit = to_tune(p_int(lower = 10,upper = 30)),
                              predict_type="prob"
)
```

```{r}
set.seed(123)
at_rpart=auto_tuner(tuner = tnr("grid_search", resolution = 3, batch_size = 3) ,
                    learner = lrn_classif.rpart_2tune,
                    resampling = rsmp("cv", folds = 3),
                    measure = msr("classif.auc"),
                    store_tuning_instance = T,
                    store_benchmark_result = T,
                    store_models = T)

```

```{r}
set.seed(123)
at_rpart$train(train_balanced)
```

```
at_rpart$tuning_instance$result
```
```{r}
set.seed(123)
pred_at_classif.rpart_train <- at_rpart$predict(train_balanced)
pred_at_classif.rpart_train$score(msr("classif.auc"))

```{r}
set.seed(123)
pred_at_classif.rpart_test <- at_rpart$predict(test_processed)
pred_at_classif.rpart_test$score(measures)
```

Create KNN learner

```{r}
lrn_kknn = lrn("classif.kknn",predict_type="prob")
lrn_kknn$param_set$default
lrn_kknn_2tune = lrn("classif.kknn",
                     k = to_tune(p_int(lower=2, upper=50)),
                     distance = to_tune(p_int(lower=1, upper=3)),
                     kernel = to_tune(c("rectangular","triangular","epanechnikov")),
                     predict_type="prob")

```

```{r}
set.seed(123)
at_knn=auto_tuner(tuner = tnr("grid_search", resolution = 3, batch_size = 3) ,
                  learner = lrn_kknn_2tune,
                  resampling = rsmp("cv", folds = 3),
                  measure = msr("classif.auc"),
                  store_tuning_instance = T,
                  store_benchmark_result = T,
                  store_models = T)
```

```{r}

```
set.seed(123)
at_knn$train(train_balanced)
at_knn$tuning_instance$result
```

```{r}
set.seed(123)
pred_at_lrn_kknn_2tune_train <- at_knn$predict(train_balanced)
pred_at_lrn_kknn_2tune_train$score(msr("classif.auc"))

```
```{r}
pred_at_lrn_kknn_2tune_test <- at_knn$predict(test_processed)
pred_at_lrn_kknn_2tune_test$score(measures)
```

Create an artificial neural network learner

```{r}
lrn_ANN=lrn("classif.nnet")
lrn_ANN$param_set
```
```{r}
lrn_nnet_2tune = lrn("classif.nnet",
                     size = to_tune(p_int(lower = 1, upper = 20)),
                     decay = to_tune(p_dbl(lower = 0.0001, upper
= 0.1, logscale = TRUE)),
                     maxit = to_tune(p_int(lower = 100, upper =
1000)),
                     predict_type = "prob")
```

```{r}
set.seed(123)
at_nnet=auto_tuner(tuner = tnr("grid_search", resolution = 3,
batch_size = 3) ,
                   learner = lrn_nnet_2tune,
                   resampling = rsmp("cv", folds = 3),
                   measure = msr("classif.auc"),
                   store_tuning_instance = T,
                   store_benchmark_result = T,
                   store_models = T)
```

```

```{r}
set.seed(123)
at_nnet$train(train_balanced)
at_nnet$tuning_instance$result
```

```{r}
pred_at_lrn_nnet_2tune_train <- at_nnet$predict(train_balanced)
pred_at_lrn_nnet_2tune_train$score(msr("classif.auc"))

```



```{r}
pred_at_lrn_nnet_2tune_test <- at_nnet$predict(test_processed)
pred_at_lrn_nnet_2tune_test$score(measures)```

### Create a random forest learner

```{r}
lrn_classif_ranger=lrn("classif.ranger")
lrn_classif_ranger$param_set
```



```{r}
lrn_ranger_2tune = lrn("classif.ranger",
                       num.trees = to_tune(p_int(lower = 50,
upper = 500)),
                       mtry = to_tune(p_int(lower = 1, upper = 7)),
```

```{r}
                       min.node.size = to_tune(p_int(lower = 1,
upper = 10)),
                       splitrule = to_tune(c("gini", "extratrees")),
                       sample.fraction = to_tune(p_dbl(0.1,1)),
                       predict_type = "prob")
```

```{r}
set.seed(123)
at_ranger=auto_tuner(tuner = tnr("grid_search", resolution = 3, batch_size = 3) ,
                     learner = lrn_ranger_2tune,
                     resampling = rsmp("cv", folds = 3),
                     measure = msr("classif.auc"),
                     store_tuning_instance = T,
                     store_benchmark_result = T,
                     store_models = T)
```

```{r}
set.seed(123)
at_ranger$train(train_balanced)
at_ranger$tuning_instance$result
```

```{r}
pred_at_lrn_ranger_2tune_train <- at_ranger$predict(train_balanced)
pred_at_lrn_ranger_2tune_train$score(msr("classif.auc"))

```

```{r}
pred_at_lrn_ranger_2tune_test <- at_ranger$predict(test_processed)
pred_at_lrn_ranger_2tune_test$score(measures)
```

Create AdaBoost learner
```{r}
lrn_adaboost=lrn("classif.AdaBoostM1")

lrn_adaboost_2tune = lrn("classif.AdaBoostM1",
                         I = to_tune(p_int(lower = 10, upper = 200)),
                         S = to_tune(p_int(lower = 1,upper = 5)),
                         predict_type = "prob")
```

```{r}
set.seed(123)
at_adaboost=auto_tuner(tuner = tnr("grid_search", resolution = 3, batch_size = 3) ,
                       learner = lrn_adaboost_2tune,
                       resampling = rsmp("cv", folds = 3),
                       measure = msr("classif.auc"),
                       store_tuning_instance = T,
                       store_benchmark_result = T,
                       store_models = T)
```

```{r}
set.seed(123)
at_adaboost$train(train_balanced)
at_adaboost$tuning_instance$result
```

```{r}
pred_at_lrn_adaboost_2tune_train <- at_adaboost$predict(train_balanced)
pred_at_lrn_adaboost_2tune_train$score(msr("classif.auc"))

```
```{r}
pred_at_lrn_adaboost_2tune_test <- at_adaboost$predict(test_processed)
pred_at_lrn_adaboost_2tune_test$score(measures)

```
Construct a gradient boosting machine
```{r}
lrn_gbm=lrn("classif.gbm")
lrn_gbm$param_set
lrn_gbm_2tune = lrn("classif.gbm",
                    n.trees = to_tune(p_int(lower = 100, upper = 500)),
                    interaction.depth = to_tune(p_int(lower = 1,upper = 10)),
                    shrinkage =to_tune(p_dbl(lower =
```

```
0.001,upper = 0.2)),
                              bag.fraction=to_tune(p_dbl(lower = 0.5,upper
= 1)),
                              n.minobsinnode=to_tune(p_int(lower =
5,upper = 50)),
                              predict_type = "prob")
```

```{r}
set.seed(123)
at_gbm=auto_tuner(tuner = tnr("grid_search", resolution = 3,
batch_size = 3) ,
                              learner = lrn_gbm_2tune,
                              resampling = rsmp("cv", folds = 3),
                              measure = msr("classif.auc"),
                              store_tuning_instance = T,
                              store_benchmark_result = T,
                              store_models = T)
```

```{r}
set.seed(123)
at_gbm$train(train_balanced)
at_gbm$tuning_instance$result
```

```{r}
pred_at_lrn_gbm_2tune_train <- at_gbm$predict(train_balanced)
pred_at_lrn_gbm_2tune_train$score(msr("classif.auc"))
```

```{r}
pred_at_lrn_gbm_2tune_test <- at_gbm$predict(test_processed)
pred_at_lrn_gbm_2tune_test$score(measures)
```

Create SVM learner

```{r}
lrn_svm = lrn("classif.svm")
lrn_svm$param_set

lrn_classif.svm2tune <-
   lrn("classif.svm",
         predict_type = "prob",
         cost    = to_tune(1e-5, 1e5, logscale = TRUE),
         gamma = to_tune(1e-5, 1e5, logscale = TRUE),
         kernel = "radial",
         type = "C-classification")

```
```{r}
set.seed(123)
at_svm=auto_tuner(tuner = tnr("grid_search", resolution = 3,
batch_size = 3) ,
                           learner = lrn_classif.svm2tune,
                           resampling = rsmp("cv", folds = 3),
                           measure = msr("classif.auc"),
                           store_tuning_instance = T,
                           store_benchmark_result = T,
                           store_models = T)
```
```{r}
set.seed(123)
at_svm$train(train_balanced)
at_svm$tuning_instance$result

```


```{r}
pred_at_lrn_svm_2tune_train <- at_svm$predict(train_balanced)
pred_at_lrn_svm_2tune_train$score(msr("classif.auc"))

```


```{r}
pred_at_svm_2tune_test <- at_svm$predict(test_processed)
pred_at_svm_2tune_test$score(measures)

```

Construct the XGBoost learner

```{r}
lrn_xgboost=lrn("classif.xgboost")
lrn_xgboost$param_set

```
```{r}
lrn_xgboost_2tune = lrn("classif.xgboost",
                        nrounds = to_tune(p_int(lower = 50, upper = 500)),
                        eta = to_tune(p_dbl(lower = 0.01,upper = 0.3,logscale = TRUE)),
                        max_depth = to_tune(p_int(lower = 3,upper = 10)),
                        min_child_weight = to_tune(p_dbl(lower = 1, upper = 10)),
                        subsample = to_tune(p_dbl(lower = 0.5, upper = 1)),
                        colsample_bytree = to_tune(p_dbl(lower = 0.5, upper = 1)),
                        lambda = to_tune(p_dbl(lower = 0, upper = 10)),
                        alpha = to_tune(p_dbl(lower = 0,upper = 10)),
                        gamma = to_tune(p_dbl(lower = 0, upper = 10)),
                        predict_type = "prob")
```
```{r}
set.seed(123)
at_xgboost=auto_tuner(tuner = tnr("grid_search", resolution = 3, batch_size = 3) ,
                      learner = lrn_xgboost_2tune,
                      resampling = rsmp("cv", folds = 3),
                      measure = msr("classif.auc"),
                      store_tuning_instance = T,
                      store_benchmark_result = T,
                      store_models = T,
                      term_evals = 50)

```

```{r}
set.seed(123)
at_xgboost$train(train_balanced)
at_xgboost$tuning_instance$result
```

```{r}
pred_at_lrn_xgboost_2tune_train <-
at_xgboost$predict(train_balanced)
autoplot(pred_at_lrn_xgboost_2tune_train,type = "roc")
```
```{r}
pred_at_xgboost_2tune_test <- at_xgboost$predict(test_processed)
pred_at_xgboost_2tune_test$score(measures)
```

Plot multiple ROC curves in a single graph

```{r}
pred_results <- list(
  "DecisionTree" = pred_at_classif.rpart_test,
  "KNN" = pred_at_lrn_kknn_2tune_test,
  "NeuralNet" = pred_at_lrn_nnet_2tune_test,
  "RandomForest" = pred_at_lrn_ranger_2tune_test,
  "AdaBoost" = pred_at_lrn_adaboost_2tune_test,
  "LightGBM" = pred_at_lrn_gbm_2tune_test,
  "SVM" = pred_at_svm_2tune_test,
  "XGBoost" = pred_at_xgboost_2tune_test
)

roc_list <- lapply(pred_results, function(df) {
  roc(as.data.table(df)$truth, as.data.table(df)$prob.1)
})
auc_values <- sapply(roc_list, auc)

performance_metrics <- data.frame(
  Model = c("XGBoost", "LightGBM", "AdaBoost", "RF", "DT", "KNN",
"NNET", "SVM"),
  AUROC = c(0.90, 0.90, 0.90, 0.88, 0.83, 0.80, 0.80, 0.75),
  ACC = c(0.91, 0.92, 0.91, 0.90, 0.90, 0.89, 0.85, 0.86),
```

```
  AUPR = c(0.74, 0.74, 0.68, 0.71, 0.69, 0.58, 0.50, 0.35),
  Brier = c(0.07, 0.08, 0.07, 0.08, 0.08, 0.09, 0.11, 0.13),
  Sensitivity = c(0.58, 0.61, 0.59, 0.50, 0.67, 0.47, 0.44, 0.02),
  Specificity = c(0.94, 0.94, 0.93, 0.92, 0.94, 0.91, 0.91, 0.86),
  PPV = c(0.77, 0.78, 0.75, 0.71, 0.67, 0.65, 0.50, 1.00),
  NPV = c(0.93, 0.94, 0.93, 0.92, 0.94, 0.91, 0.91, 0.86),
  F1 = c(0.66, 0.68, 0.66, 0.59, 0.67, 0.55, 0.46, 0.03),
  stringsAsFactors = FALSE
)

performance_metrics <- performance_metrics %>%
  arrange(desc(AUROC), desc(AUPR))
roc_data <- do.call(rbind, lapply(seq_along(roc_list), function(i) {
  model_name <- names(roc_list)[i]
  data.frame(
    TPR = roc_list[[i]]$sensitivities,
    FPR = 1 - roc_list[[i]]$specificities,
    Model = model_name,
    stringsAsFactors = FALSE
  )
}))

name_mapping <- c(
  "DecisionTree" = "DT",
  "KNN" = "KNN",
  "NeuralNet" = "NNET",
  "RandomForest" = "RF",
  "AdaBoost" = "AdaBoost",
  "LightGBM" = "LightGBM",
  "SVM" = "SVM",
  "XGBoost" = "XGBoost"
)

roc_data$Model <- name_mapping[roc_data$Model]
roc_data$Model <- factor(roc_data$Model, levels =
performance_metrics$Model)

model_colors <- c(
  "XGBoost" = "#00468B",
  "LightGBM" = "#0099B4",
  "AdaBoost" = "#42B540",
  "RF" = "#ADB6B6",
```

```
  "DT" = "#E69F00",
  "KNN" = "#F0E442",
  "NNET" = "#D55E00",
  "SVM" = "#CC79A7"
)

legend_labels <- setNames(
  paste0(performance_metrics$Model, " (", sprintf("%.2f",
performance_metrics$AUROC), ")"),
  performance_metrics$Model
)

p_roc <- ggplot(roc_data, aes(x = FPR, y = TPR, color = Model)) +
  geom_line(linewidth = 0.8) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed", color =
"gray60") +
  scale_color_manual(
    values = model_colors,
    labels = legend_labels,
    breaks = performance_metrics$Model,
    limits = performance_metrics$Model
  ) +
  coord_equal(xlim = c(0, 1), ylim = c(0, 1)) +
  labs(
    x = "False Positive Rate (1 - Specificity)",
    y = "True Positive Rate (Sensitivity)"
  ) +
  theme_minimal(base_size = 11) +
  theme(
    panel.background = element_rect(fill = "white", color = NA),
    plot.background = element_rect(fill = "white", color = NA),
    legend.key = element_rect(fill = "white", color = NA),
    axis.line = element_line(color = "black", linewidth = 0.5),
    legend.title = element_blank(),
    legend.text = element_text(size = 7),
    legend.position = "right",
    legend.background = element_rect(fill = "white", color = NA),
    legend.margin = margin(2, 2, 2, 2),
    legend.spacing.y = unit(1, "mm"),
    axis.title = element_text(size = 11),
    axis.text = element_text(size = 10),
    panel.grid.major = element_blank(),
```

```
        panel.grid.minor = element_blank()
    ) +
    guides(color = guide_legend(
        ncol = 1,
        byrow = TRUE,
        keyheight = unit(3, "mm"),
        keywidth = unit(3.5, "mm")
    ))

table_data <- performance_metrics %>%
    mutate(
        AUROC = sprintf("%.2f", AUROC),
        AUPR = sprintf("%.2f", AUPR),
        Brier = sprintf("%.2f", Brier),
        Sens = sprintf("%.2f", Sensitivity),
        Spec = sprintf("%.2f", Specificity),
        PPV = sprintf("%.2f", PPV),
        NPV = sprintf("%.2f", NPV),
        F1 = sprintf("%.2f", F1)
    ) %>%
    select(Model, AUROC, AUPR, Brier, Sens, Spec, PPV, NPV, F1)

table_grob <- tableGrob(
    table_data,
    rows = NULL,
    theme = ttheme_default(
        core = list(
            fg_params = list(fontsize = 8, fontface = "plain"),
            bg_params = list(
                fill = c(rep("#E8F4F8", 4), rep("white", 4)),
                col = "gray70",
                lwd = 0.5
            ),
            padding = unit(c(1.5, 3), "mm")
        ),
        colhead = list(
            fg_params = list(fontsize = 8, fontface = "bold"),
            bg_params = list(fill = "#B8D4E3", col = "gray70", lwd = 0.5),
            padding = unit(c(2, 3), "mm")
        )
    )
)
```

```
table_note <- textGrob(
  "Top 4 models (shaded): AUROC 0.88–0.90, no significant difference
by DeLong's test (all P > 0.05)",
  gp = gpar(fontsize = 7, lineheight = 1.2, col = "gray40"),
  just = "center"
)

table_layout <- gtable(
  widths = unit(1, "npc"),
  heights = unit(c(0.85, 0.15), c("npc", "npc"))
)
table_layout <- gtable_add_grob(table_layout, table_grob, t = 1, l = 1)
table_layout <- gtable_add_grob(table_layout, table_note, t = 2, l = 1)

p_table <- ggplot() +
  annotation_custom(table_layout, xmin = 0, xmax = 1, ymin = 0, ymax
= 1) +
  theme_void() +
  theme(plot.margin = margin(5, 5, 5, 5))
p_roc_tagged <- p_roc + labs(tag = "A")
p_table_tagged <- p_table + labs(tag = "B")
p_combined <- p_roc_tagged / p_table_tagged +
  plot_layout(heights = c(2, 1))
ggsave(
  filename = "NoteFixed_600dpi.png",
  plot = p_combined,
  dpi = 600,
  width = 18,
  height = 22,
  units = "cm",
  bg = "white"
)
```

Step 4：DeLong's Test for AUROC Comparison

```{r}
true_labels <- test_processed$truth()
pred_probs <- list(
  DT          = pred_at_classif.rpart_test$prob[, "1"],
  KNN         = pred_at_lrn_kknn_2tune_test$prob[, "1"],
  NNET        = pred_at_lrn_nnet_2tune_test$prob[, "1"],
```

```
  RF           = pred_at_lrn_ranger_2tune_test$prob[, "1"],
  AdaBoost= pred_at_lrn_adaboost_2tune_test$prob[, "1"],
  LightGBM= pred_at_lrn_gbm_2tune_test$prob[, "1"],
  SVM          = pred_at_svm_2tune_test$prob[, "1"],
  XGBoost = pred_at_xgboost_2tune_test$prob[, "1"]
)
str(pred_probs)
roc_list <- lapply(names(pred_probs), function(model_name) {
  roc(response = true_labels,
      predictor = pred_probs[[model_name]],
      levels = c("0", "1"),
      direction = "<",
      quiet = TRUE)
})

names(roc_list) <- names(pred_probs)
cat("===AUROC ===\n")
for (name in names(roc_list)) {
  cat(sprintf("%-10s: %.4f (95%% CI: %.4f-%.4f)\n",
              name,
              roc_list[[name]]$auc,
              ci.auc(roc_list[[name]])[1],
              ci.auc(roc_list[[name]])[3]))
}
cat("\n=== DeLong's Test: XGBoost vs Other Models ===\n")

xgb_roc <- roc_list[["XGBoost"]]
delong_results <- data.frame(
  Model = character(),
  AUROC = numeric(),
  XGBoost_AUROC = numeric(),
  Difference = numeric(),
  Z_statistic = numeric(),
  P_value = numeric(),
  Significant = character(),
  stringsAsFactors = FALSE
)

for (name in setdiff(names(roc_list), "XGBoost")) {
  test_result <- roc.test(xgb_roc, roc_list[[name]], method = "delong")

  delong_results <- rbind(delong_results, data.frame(
```

```
      Model = name,
      AUROC = as.numeric(roc_list[[name]]$auc),
      XGBoost_AUROC = as.numeric(xgb_roc$auc),
      Difference = as.numeric(xgb_roc$auc) -
as.numeric(roc_list[[name]]$auc),
      Z_statistic = as.numeric(test_result$statistic),
      P_value = as.numeric(test_result$p.value),
      Significant = ifelse(test_result$p.value < 0.05, "Yes", "No")
   ))
}

print(delong_results)
model_names <- names(roc_list)
n_models <- length(model_names)

p_value_matrix <- matrix(NA, n_models, n_models,
                         dimnames = list(model_names,
model_names))
diff_matrix <- matrix(NA, n_models, n_models,
                      dimnames = list(model_names,
model_names))

for (i in 1:(n_models-1)) {
   for (j in (i+1):n_models) {
      test_result <- roc.test(roc_list[[i]], roc_list[[j]], method = "delong")
      p_value_matrix[i, j] <- test_result$p.value
      p_value_matrix[j, i] <- test_result$p.value
      diff_matrix[i, j] <- as.numeric(roc_list[[i]]$auc) -
as.numeric(roc_list[[j]]$auc)
      diff_matrix[j, i] <- -diff_matrix[i, j]
   }
}
print(round(diff_matrix, 4))
n_comparisons <- n_models * (n_models - 1) / 2
alpha_corrected <- 0.05 / n_comparisons

sig_matrix <- p_value_matrix < alpha_corrected
sig_matrix[is.na(sig_matrix)] <- FALSE
print(sig_matrix)
auc_ci <- lapply(roc_list, function(r) {
   ci <- ci.auc(r)
   data.frame(
```

```
        AUROC = as.numeric(r$auc),
        Lower = ci[1],
        Upper = ci[3]
    )
})

auc_df <- do.call(rbind, auc_ci)
auc_df$Model <- rownames(auc_df)
auc_df <- auc_df[order(-auc_df$AUROC), ]
auc_df$Model <- factor(auc_df$Model, levels = auc_df$Model)

p1 <- ggplot(auc_df, aes(x = AUROC, y = Model)) +
    geom_point(size = 3, color = "steelblue") +
    geom_errorbarh(aes(xmin = Lower, xmax = Upper), height = 0.2,
color = "steelblue") +
    geom_vline(xintercept = 0.5, linetype = "dashed", color = "red",
alpha = 0.5) +
    scale_x_continuous(limits = c(0.5, 1.0), breaks = seq(0.5, 1.0, 0.1)) +
    labs(title = "AUROC with 95% Confidence Intervals",
            x = "AUROC", y = "") +
    theme_minimal() +
    theme(plot.title = element_text(hjust = 0.5, face = "bold"))

print(p1)
ggsave("AUROC_forest_plot.png", p1, width = 8, height = 6, dpi = 600)
library(reshape2)

p_melt <- melt(p_value_matrix, na.rm = TRUE)
p_melt$value_label <- ifelse(p_melt$value < 0.001, "<0.001",
sprintf("%.3f", p_melt$value))

p2 <- ggplot(p_melt, aes(Var1, Var2, fill = value)) +
    geom_tile(color = "white") +
    geom_text(aes(label = value_label), size = 3) +
    scale_fill_gradient2(low = "red", mid = "yellow", high = "green",
                                    midpoint = 0.05, limits = c(0, 1),
                                    name = "P-value") +
    labs(title = "DeLong's Test P-value Matrix",
            x = "", y = "") +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1),
              plot.title = element_text(hjust = 0.5, face = "bold"))
```

```
print(p2)
ggsave("Delong_pvalue_heatmap.png", p2, width = 10, height = 8, dpi
= 600)
```

```
## Step 5: Simplify the model

```{r}
df10 <- read.xlsx("Data10.xlsx") %>%
  select(-id) %>%
  mutate(across(where(is.character), as.factor))
df10$Nephrotoxicity <- factor(ifelse(df10$Nephrotoxicity == "yes",
"1", "0"),
                              levels = c("0", "1"))
str(df10)
```


```{r}
task10 <- as_task_classif(x=df10,target = "Nephrotoxicity",id="test"
                          ,positive = "1")
global_pipeline <- po("imputehist") %>>%
  po("imputemode") %>>%
  po("scale", affect_columns = selector_type("numeric")) %>>%
  po("encode",method="treatment",affect_columns =
selector_type("factor"))
set.seed(123)
split <- partition(task10, ratio = 0.7)
train_task_10 <- task10$clone()$filter(split$train)
test_task_10 <- task10$clone()$filter(split$test)
global_pipeline$train(train_task_10)
train_processed_10 <- global_pipeline$predict(train_task_10)[[1]]
test_processed_10 <- global_pipeline$predict(test_task_10)[[1]]
set.seed(123)
smote_pipe <- po("smote", dup_size = 2)
train_balanced_10 <- smote_pipe$train(list(train_processed_10))[[1]]
measures = list(
  msr("classif.auc"),
  msr("classif.acc"),
  msr("classif.prauc"),
  msr("classif.bbrier"),
```
```

```
  msr("classif.sensitivity"),
  msr("classif.ppv"),
  msr("classif.npv"),
  msr("classif.fbeta")
)
lrn_xgboost_10=lrn("classif.xgboost")

```
```{r}
lrn_xgboost_2tune_10 = lrn("classif.xgboost",
                           nrounds = to_tune(p_int(lower = 50, upper = 500)),
                           eta = to_tune(p_dbl(lower = 0.01,upper = 0.3,logscale = TRUE)),
                           max_depth = to_tune(p_int(lower = 3,upper = 10)),
                           min_child_weight = to_tune(p_dbl(lower = 1, upper = 10)),
                           subsample = to_tune(p_dbl(lower = 0.5, upper = 1)),
                           colsample_bytree = to_tune(p_dbl(lower = 0.5, upper = 1)),
                           lambda = to_tune(p_dbl(lower = 0, upper = 10)),
                           alpha = to_tune(p_dbl(lower = 0,upper = 10)),
                           gamma = to_tune(p_dbl(lower = 0, upper = 10)),
                           predict_type = "prob")
```
```{r}
set.seed(123)
at_xgboost_10=auto_tuner(tuner = tnr("grid_search", resolution = 3, batch_size = 3) ,
                         learner = lrn_xgboost_2tune_10,
                         resampling = rsmp("cv", folds = 3),
                         measure = msr("classif.auc"),
                         store_tuning_instance = T,
                         store_benchmark_result = T,
                         store_models = T,
                         term_evals = 50)
```

```
```{r}
set.seed(123)
at_xgboost_10$train(train_balanced_10)

```
```{r}
pred_at_lrn_xgboost_2tune_train_10 <-
at_xgboost_10$predict(train_balanced_10)
pred_at_lrn_xgboost_2tune_train_10$score(msr("classif.auc"))

```
```{r}
pred_at_xgboost_2tune_test_10 <-
at_xgboost_10$predict(test_processed_10)
pred_at_xgboost_2tune_test_10$score(measures)
```
```{r}
df8 <- read.xlsx("Data8.xlsx") %>%
  select(-id) %>%
  mutate(across(where(is.character), as.factor))
df8$Nephrotoxicity <- factor(ifelse(df8$Nephrotoxicity == "yes", "1",
"0"),
                             levels = c("0", "1"))
```


```{r}
task_8 <- as_task_classif(x=df8,target = "Nephrotoxicity",
                          id="test",positive = "1")
global_pipeline <- po("imputehist") %>>%
  po("imputemode") %>>%
  po("scale", affect_columns = selector_type("numeric")) %>>%
  po("encode",method="treatment",affect_columns =
selector_type("factor"))
set.seed(123)
split <- partition(task_8, ratio = 0.7)
train_task_8 <- task_8$clone()$filter(split$train)
test_task_8 <- task_8$clone()$filter(split$test)
global_pipeline$train(train_task_8)
train_processed_8 <- global_pipeline$predict(train_task_8)[[1]]
test_processed_8 <- global_pipeline$predict(test_task_8)[[1]]
set.seed(123)
```
```

```
smote_pipe <- po("smote", dup_size = 2)
train_balanced_8 <- smote_pipe$train(list(train_processed_8))[[1]]
measures = list(
    msr("classif.auc"),
    msr("classif.acc"),
    msr("classif.prauc"),
    msr("classif.bbrier"),
    msr("classif.sensitivity"),
    msr("classif.ppv"),
    msr("classif.npv"),
    msr("classif.fbeta")
)
lrn_xgboost_8=lrn("classif.xgboost")

```

```{r}
lrn_xgboost_2tune_8 = lrn("classif.xgboost",
                          nrounds = to_tune(p_int(lower = 50, upper = 500)),
                          eta = to_tune(p_dbl(lower = 0.01,upper = 0.3,logscale = TRUE)),
                          max_depth = to_tune(p_int(lower = 3,upper = 10)),
                          min_child_weight = to_tune(p_dbl(lower = 1, upper = 10)),
                          subsample = to_tune(p_dbl(lower = 0.5, upper = 1)),
                          colsample_bytree = to_tune(p_dbl(lower = 0.5, upper = 1)),
                          lambda = to_tune(p_dbl(lower = 0, upper = 10)),
                          alpha = to_tune(p_dbl(lower = 0,upper = 10)),
                          gamma = to_tune(p_dbl(lower = 0, upper = 10)),
                          predict_type = "prob")
```
```{r}
set.seed(123)
at_xgboost_8=auto_tuner(tuner = tnr("grid_search", resolution = 3, batch_size = 3) ,
```

```
                                learner = lrn_xgboost_2tune_8,
                                resampling = rsmp("cv", folds = 3),
                                measure = msr("classif.auc"),
                                store_tuning_instance = T,
                                store_benchmark_result = T,
                                store_models = T,
                                term_evals = 50)
```

```{r}
set.seed(123)
at_xgboost_8$train(train_balanced_8)
```

```{r}
pred_at_lrn_xgboost_2tune_train_8 <-
at_xgboost_8$predict(train_balanced_8)
pred_at_lrn_xgboost_2tune_train_8$score(msr("classif.auc"))
```

```{r}
pred_at_xgboost_2tune_test_8 <-
at_xgboost_8$predict(test_processed_8)
pred_at_xgboost_2tune_test_8$score(measures)
```
```{r}
df6 <- read.xlsx("Data6.xlsx") %>%
  select(-id) %>%
  mutate(across(where(is.character), as.factor))
df6$Nephrotoxicity <- factor(ifelse(df6$Nephrotoxicity == "yes", "1",
"0"),
                             levels = c("0", "1"))
```

```{r}
task_6 <- as_task_classif(x=df6,target = "Nephrotoxicity",
                          id="test",positive = "1")
global_pipeline <- po("imputehist") %>>%
  po("imputemode") %>>%
```

```
    po("scale", affect_columns = selector_type("numeric")) %>>%
    po("encode",method="treatment",affect_columns =
selector_type("factor"))
set.seed(123)
split <- partition(task_6, ratio = 0.7)
train_task_6 <- task_6$clone()$filter(split$train)
test_task_6 <- task_6$clone()$filter(split$test)
global_pipeline$train(train_task_6)
train_processed_6 <- global_pipeline$predict(train_task_6)[[1]]
test_processed_6 <- global_pipeline$predict(test_task_6)[[1]]
set.seed(123)
smote_pipe <- po("smote", dup_size = 2)
train_balanced_6 <- smote_pipe$train(list(train_processed_6))[[1]]
measures = list(
    msr("classif.auc"),
    msr("classif.acc"),
    msr("classif.prauc"),
    msr("classif.bbrier"),
    msr("classif.sensitivity"),
    msr("classif.ppv"),
    msr("classif.npv"),
    msr("classif.fbeta")
)
lrn_xgboost_6=lrn("classif.xgboost")

```
```{r}
lrn_xgboost_2tune_6 = lrn("classif.xgboost",
                          nrounds = to_tune(p_int(lower = 50,
upper = 500)),
                          eta = to_tune(p_dbl(lower = 0.01,upper
= 0.3,logscale = TRUE)),
                          max_depth = to_tune(p_int(lower =
3,upper = 10)),
                          min_child_weight =
to_tune(p_dbl(lower = 1, upper = 10)),
                          subsample = to_tune(p_dbl(lower = 0.5,
upper = 1)),
                          colsample_bytree =
to_tune(p_dbl(lower = 0.5, upper = 1)),
                          lambda = to_tune(p_dbl(lower = 0,
upper = 10)),
```

```
                                    alpha = to_tune(p_dbl(lower = 0,upper
= 10)),
                                    gamma = to_tune(p_dbl(lower = 0,
upper = 10)),
                                    predict_type = "prob")
```
```{r}
set.seed(123)
at_xgboost_6=auto_tuner(tuner = tnr("grid_search", resolution = 3,
batch_size = 3) ,
                                learner = lrn_xgboost_2tune_6,
                                resampling = rsmp("cv", folds = 3),
                                measure = msr("classif.auc"),
                                store_tuning_instance = T,
                                store_benchmark_result = T,
                                store_models = T,
                                term_evals = 50)

```
```{r}
set.seed(123)
at_xgboost_6$train(train_balanced_6)

```
```{r}
pred_at_lrn_xgboost_2tune_train_6                                    <-
at_xgboost_6$predict(train_balanced_6)
pred_at_lrn_xgboost_2tune_train_6$score(msr("classif.auc"))

```


```{r}
pred_at_xgboost_2tune_test_6                                         <-
at_xgboost_6$predict(test_processed_6)

pred_at_xgboost_2tune_test_6$score(measures)
```
```{r}
pred_list <- list(
   full = pred_at_xgboost_2tune_test,
   feat10 = pred_at_xgboost_2tune_test_10,
   feat8 = pred_at_xgboost_2tune_test_8,
```

```
    feat6 = pred_at_xgboost_2tune_test_6
)
head(pred_list$feat10$prob)
true_label <- as.numeric(as.character(pred_list$full$truth))
prob_full    <- pred_list$full$prob[, "1"]
prob_10        <- pred_list$feat10$prob[, "1"]
prob_8         <- pred_list$feat8$prob[, "1"]
prob_6         <- pred_list$feat6$prob[, "1"]

roc_full    <- roc(true_label, prob_full, quiet = TRUE)
roc_10        <- roc(true_label, prob_10, quiet = TRUE)
roc_8         <- roc(true_label, prob_8, quiet = TRUE)
roc_6         <- roc(true_label, prob_6, quiet = TRUE)
auc_results <- data.frame(
    Model = c("Full Model", "10 Features", "8 Features", "6 Features"),
    AUC = c(auc(roc_full), auc(roc_10), auc(roc_8), auc(roc_6)),
    CI_lower = c(ci(roc_full)[1], ci(roc_10)[1], ci(roc_8)[1], ci(roc_6)[1]),
    CI_upper = c(ci(roc_full)[3], ci(roc_10)[3], ci(roc_8)[3], ci(roc_6)[3])
)
print(auc_results)
comparisons <- list(
    list("Full vs 10 Features", roc_full, roc_10),
    list("Full vs 8 Features",    roc_full, roc_8),
    list("Full vs 6 Features",    roc_full, roc_6),
    list("10 vs 8 Features",       roc_10, roc_8),
    list("10 vs 6 Features",       roc_10, roc_6),
    list("8 vs 6 Features",         roc_8, roc_6)
)
delong_results <- data.frame(
    Comparison = character(),
    AUC_Diff = numeric(),
    Z_value = numeric(),
    P_value = numeric(),
    Significance = character(),
    stringsAsFactors = FALSE
)

for(i in seq_along(comparisons)) {
    comp_name <- comparisons[[i]][[1]]
    roc1 <- comparisons[[i]][[2]]
    roc2 <- comparisons[[i]][[3]]
    test <- roc.test(roc1, roc2, method = "delong")
```

```
  auc_diff <- auc(roc1) - auc(roc2)
  z_val <- test$statistic
  p_val <- test$p.value
  sig <- ifelse(p_val < 0.001, "***",
                ifelse(p_val < 0.01, "**",
                       ifelse(p_val < 0.05, "*", "ns")))

  delong_results <- rbind(delong_results, data.frame(
    Comparison = comp_name,
    AUC_Diff = round(auc_diff, 4),
    Z_value = round(as.numeric(z_val), 4),
    P_value = format(p_val, scientific = TRUE, digits = 4),
    Significance = sig
  ))
}

print(delong_results)
plot_data <- data.frame(
  Model = factor(c("Full Features", "10 Features", "8 Features", "6 Features"),
                 levels = c("6 Features", "8 Features", "10 Features", "Full Features")),
  AUC = auc_results$AUC,
  CI_lower = auc_results$CI_lower,
  CI_upper = auc_results$CI_upper
)

p <- ggplot(plot_data, aes(x = Model, y = AUC, fill = Model)) +
  geom_bar(stat = "identity", width = 0.6, color = "black", linewidth = 0.3) +
  geom_errorbar(aes(ymin = CI_lower, ymax = CI_upper),
                width = 0.2, linewidth = 0.8, color = "black") +
  geom_text(aes(label = sprintf("%.4f", AUC)),
            nudge_y = 0.08, size = 4, fontface = "bold") +
  geom_text(aes(label = sprintf("[%.3f, %.3f]", CI_lower, CI_upper)),
            nudge_y = -0.06, size = 3, color = "white") +
  scale_fill_manual(values = c("Full Features" = "#E41A1C",
                               "10 Features" = "#377EB8",
                               "8 Features" = "#4DAF4A",
                               "6 Features" = "#FF7F00")) +
  coord_cartesian(ylim = c(0, 1.05)) +
  labs(
```

```
        title = "AUC Comparison Across Feature Sets",
        y = "AUC (95% CI)",
        caption = "Error bars represent 95% confidence intervals"
    ) +
    theme_minimal(base_size = 12) +
    theme(
        legend.position = "none",
        axis.title.x = element_blank(),
        plot.title = element_text(face = "bold", size = 14, hjust = 0.5),
        plot.subtitle = element_text(size = 11, hjust = 0.5, color = "grey30"),
        axis.text = element_text(size = 11),
        panel.grid.major.x = element_blank()
    )
p
png("AUC_Comparison_Delong.png", width = 4000, height = 3500,
        units = "px", res = 600, type = "cairo", bg = "white")
print(p)
dev.off()

```
Step 6: Confusion Matrix Calculation
truth_labels <- as.character(pred_at_xgboost_2tune_test_6$truth)
pred_probs <- pred_at_xgboost_2tune_test_6$prob[, "1"]
```{r}
roc_obj <- roc(truth_labels, pred_probs)
best_coords <- coords(roc_obj, "best", best.method = "youden", transpose = FALSE)
youden_threshold <- best_coords$threshold[1]

thresholds_to_test <- c(0.05, 0.10, 0.142, 0.20, 0.30, 0.50, youden_threshold)

results_list <- list()

for (thresh in thresholds_to_test) {
    pred_lbl <- ifelse(pred_probs >= thresh, "1", "0")
    cm <- table(Predicted = pred_lbl, Observed = truth_labels)

    TP <- cm["1","1"]; FN <- cm["0","1"]
    TN <- cm["0","0"]; FP <- cm["1","0"]
```

```
    sens <- TP/(TP+FN)
    spec <- TN/(TN+FP)
    ppv <- TP/(TP+FP)
    npv <- TN/(TN+FN)

    results_list[[length(results_list)+1]] <- data.frame(
        Threshold = round(thresh, 4),
        TP = TP, FP = FP, TN = TN, FN = FN,
        Sensitivity = round(sens, 3),
        Specificity = round(spec, 3),
        PPV = round(ppv, 3),
        NPV = round(npv, 3),
        Youden = round(sens + spec - 1, 3)
    )
}

results_df <- do.call(rbind, results_list)
print(results_df[order(results_df$Youden, decreasing = TRUE), ])
```

Step 7: Plot the Youden index versus threshold curve

```{r}
youden_data <- data.frame(
    Threshold = thresholds_to_test,
    Youden = results_df$Youden,
    Sensitivity = results_df$Sensitivity,
    Specificity = results_df$Specificity
)

youden_plot <- ggplot(youden_data, aes(x = Threshold)) +
    geom_line(aes(y = Youden, color = "Youden Index"), linewidth = 1.2)
+
    geom_line(aes(y = Sensitivity, color = "Sensitivity"), linewidth = 0.8,
linetype = "dashed") +
    geom_line(aes(y = Specificity, color = "Specificity"), linewidth = 0.8,
linetype = "dashed") +
    geom_point(aes(y = Youden), data =
youden_data[youden_data$Threshold == 0.3978, ],
                size = 4, color = "red") +
    annotate("text", x = 0.3978, y = 0.72,
             label = "Optimal\n(0.3978)", color = "red", size = 3.5,
```

```
hjust = -0.2) +
  scale_color_manual(values = c("Youden Index" = "black",
                                "Sensitivity" = "#E41A1C",
                                "Specificity" = "#377EB8")) +
  labs(x = "Threshold", y = "Value",
       title = "Figure S9. Threshold Optimization via Youden Index",
       subtitle = "Optimal threshold = 0.3978 (Youden = 0.694)") +
  theme_minimal(base_size = 11) +
  theme(legend.position = "top", plot.title = element_text(face = "bold"))

ggsave("Figure_S7_Threshold_Optimization.png", youden_plot, width = 8, height = 5, dpi = 600)
```

```{r}

```

Step 8：Nested CV + Bootstrap Optimism Correction

```
main_test_auc <- pred_at_xgboost_2tune_test_6$score(msr("classif.auc"))
main_test_prauc <- pred_at_xgboost_2tune_test_6$score(msr("classif.prauc"))
main_test_brier <- pred_at_xgboost_2tune_test_6$score(msr("classif.bbrier"))
```

```{r}
```

```
set.seed(123)
n_repeats <- 3
n_outer_folds <- 5
nested_cv_results <- list()
for (rep in 1:n_repeats) {
  cat("\n=== Repeat", rep, "of", n_repeats, "===\n")
  set.seed(123 + rep * 100)
  all_ids <- task_6$row_ids
  all_data <- task_6$data()
  pos_ids <- all_ids[all_data$Nephrotoxicity == "1"]
  neg_ids <- all_ids[all_data$Nephrotoxicity == "0"]
  set.seed(123 + rep * 100)
  pos_shuffled <- sample(pos_ids)
  neg_shuffled <- sample(neg_ids)
```

```
  pos_fold_size <- floor(length(pos_shuffled) / n_outer_folds)
  neg_fold_size <- floor(length(neg_shuffled) / n_outer_folds)

  outer_folds <- list()
  for (fold in 1:n_outer_folds) {
    if (fold < n_outer_folds) {
      pos_test <- pos_shuffled[((fold-1)*pos_fold_size +
1):(fold*pos_fold_size)]
      neg_test <- neg_shuffled[((fold-1)*neg_fold_size +
1):(fold*neg_fold_size)]
    } else {
      pos_test <- pos_shuffled[((fold-1)*pos_fold_size +
1):length(pos_shuffled)]
      neg_test <- neg_shuffled[((fold-1)*neg_fold_size +
1):length(neg_shuffled)]
    }
    test_ids <- c(pos_test, neg_test)
    train_ids <- setdiff(all_ids, test_ids)
    outer_folds[[fold]] <- list(train = train_ids, test = test_ids)
  }

  for (fold in 1:n_outer_folds) {

    cat("    Outer fold", fold, "of", n_outer_folds, "\n")

    outer_train_ids <- outer_folds[[fold]]$train
    outer_test_ids <- outer_folds[[fold]]$test
    outer_train_task <- task_6$clone()$filter(outer_train_ids)
    outer_test_task <- task_6$clone()$filter(outer_test_ids)
    outer_pipeline <- po("imputehist") %>>%
      po("imputemode") %>>%
      po("scale", affect_columns = selector_type("numeric")) %>>%
      po("encode", method = "treatment", affect_columns =
selector_type("factor"))
    outer_pipeline$train(outer_train_task)
    outer_train_processed <-
outer_pipeline$predict(outer_train_task)[[1]]
    outer_test_processed <-
outer_pipeline$predict(outer_test_task)[[1]]
    outer_smote <- po("smote", dup_size = 2)
    set.seed(123 + rep * 100 + fold)
    outer_train_balanced <-
```

```
outer_smote$train(list(outer_train_processed))[[1]]
      inner_lrn <- lrn("classif.xgboost",
                              nrounds = to_tune(p_int(lower = 100, upper
= 500)),
                              eta = to_tune(p_dbl(lower = 0.01, upper = 0.3,
logscale = TRUE)),
                              max_depth = to_tune(p_int(lower = 3, upper =
10)),
                              min_child_weight = to_tune(p_dbl(lower = 1,
upper = 10)),
                              subsample = to_tune(p_dbl(lower = 0.6,
upper = 1.0)),
                              colsample_bytree = to_tune(p_dbl(lower =
0.6, upper = 1.0)),
                              lambda = to_tune(p_dbl(lower = 0, upper =
10)),
                              alpha = to_tune(p_dbl(lower = 0, upper = 10)),
                              gamma = to_tune(p_dbl(lower = 0, upper =
10)),
                              predict_type = "prob")
      inner_at <- auto_tuner(
         tuner = tnr("random_search", batch_size = 5),
         learner = inner_lrn,
         resampling = rsmp("cv", folds = 3),
         measure = msr("classif.auc"),
         term_evals = 50
      )

      set.seed(123 + rep * 100 + fold)
      inner_at$train(outer_train_balanced)
      outer_pred <- inner_at$predict(outer_test_processed)
      auc_value <- outer_pred$score(msr("classif.auc"))
      acc_value <- outer_pred$score(msr("classif.acc"))
      sens_value <- outer_pred$score(msr("classif.sensitivity"))
      spec_value <- outer_pred$score(msr("classif.specificity"))
      prauc_value <- outer_pred$score(msr("classif.prauc"))
      brier_value <- outer_pred$score(msr("classif.bbrier"))

      cat(sprintf("       AUROC = %.3f | Sensitivity = %.3f | Specificity
= %.3f\n",
                  auc_value, sens_value, spec_value))
```

```
        nested_cv_results[[length(nested_cv_results) + 1]] <- data.frame(
            Repeat = rep,
            Outer_Fold = fold,
            AUROC = auc_value,
            AUPR = prauc_value,
            Brier = brier_value,
            Accuracy = acc_value,
            Sensitivity = sens_value,
            Specificity = spec_value
        )
    }
}
nested_cv_df <- do.call(rbind, nested_cv_results)

cat("\n=== Nested CV Summary ===\n")
cat("Total evaluations:", nrow(nested_cv_df), "\n")
cat("AUROC range:", sprintf("%.3f", min(nested_cv_df$AUROC)), "-",
        sprintf("%.3f", max(nested_cv_df$AUROC)), "\n")
cat("Mean AUROC:", sprintf("%.3f", mean(nested_cv_df$AUROC)),
"\n")
cat("SD AUROC:", sprintf("%.3f", sd(nested_cv_df$AUROC)), "\n")
cat("Any NA:", any(is.na(nested_cv_df$AUROC)), "\n")

# Table S6 - By Repeat
table_s6_by_repeat <- nested_cv_df %>%
    group_by(Repeat) %>%
    summarise(
        AUROC_mean = mean(AUROC),
        AUROC_sd = sd(AUROC),
        AUROC_min = min(AUROC),
        AUROC_max = max(AUROC),
        Sensitivity_mean = mean(Sensitivity),
        Specificity_mean = mean(Specificity),
        Brier_mean = mean(Brier),
        .groups = "drop"
    )

# Table S6 - Overall
table_s6_overall <- nested_cv_df %>%
    summarise(
        AUROC_mean = mean(AUROC),
        AUROC_sd = sd(AUROC),
```

```
    AUROC_95CI_lower = quantile(AUROC, 0.025),
    AUROC_95CI_upper = quantile(AUROC, 0.975),
    Sensitivity_mean = mean(Sensitivity),
    Specificity_mean = mean(Specificity),
    Brier_mean = mean(Brier),
    N_folds = n(),
    .groups = "drop"
  )

cat("\n=== Table S6: Nested CV Results by Repeat ===\n")
print(table_s6_by_repeat)
cat("\n=== Table S6: Nested CV Overall Summary ===\n")
print(table_s6_overall)

write.csv(table_s6_by_repeat, "Table_S9_Nested_CV_By_Repeat.csv",
row.names = FALSE)
write.csv(table_s6_overall, "Table_S9_Nested_CV_Overall.csv",
row.names = FALSE)

if (nrow(nested_cv_df) > 0 && !any(is.na(nested_cv_df$AUROC))) {
  y_min <- floor(min(nested_cv_df$AUROC) * 20) / 20
  y_max <- ceiling(max(nested_cv_df$AUROC) * 20) / 20 + 0.02

  figure_s8 <- ggplot(nested_cv_df, aes(x = factor(Outer_Fold), y =
AUROC,
                                        fill = factor(Repeat)))
+
    geom_bar(stat = "identity", position = position_dodge(width =
0.7),
             width = 0.6, color = "black", linewidth = 0.3) +
    geom_text(aes(label = sprintf("%.3f", AUROC)),
              position = position_dodge(width = 0.7),
              vjust = -0.3,
              size = 3,
              fontface = "bold",
              color = "black") +
    geom_hline(yintercept = main_test_auc, linetype = "dashed",
               color = "red", linewidth = 1) +
    annotate("text", x = 3, y = max(nested_cv_df$AUROC) + 0.015,
             label = paste("Main analysis AUROC =",
round(main_test_auc, 3)),
             color = "red", size = 4, fontface = "bold") +
```

```
      scale_y_continuous(limits = c(y_min, y_max),
                                 breaks = seq(y_min, y_max, 0.02),
                                 oob = scales::squish,
                                 expand = expansion(mult = c(0, 0.05))) +
      scale_fill_brewer(palette = "Set2", name = "Repeat") +
      labs(x = "Outer CV Fold", y = "AUROC",
           title = "Figure S7 Repeated Nested Cross-Validation Results
(6-Feature Model)",
           subtitle = "5-fold outer CV × 3 repeats (15 independent
evaluations)") +
      theme_minimal(base_size = 12) +
      theme(
        plot.title = element_text(size = 14, face = "bold"),
        plot.subtitle = element_text(size = 11, color = "gray40"),
        legend.position = "top",
        panel.border = element_rect(color = "black", fill = NA, linewidth
= 0.5),
        axis.text.x = element_text(size = 11),
        axis.text.y = element_text(size = 11),
        panel.grid.major.y = element_line(color = "gray90"),
        panel.grid.minor.y = element_line(color = "gray95")
      )

  ggsave("Figure_S8_Nested_CV_Results_6feature.png", figure_s8,
           width = 12, height = 7, dpi = 300)
  cat("\nFigure S8 saved successfully\n")
  print(figure_s8)

} else {
  cat("\nWARNING: Cannot create Figure S8 - data issues detected\n")
}

# BOOTSTRAP OPTIMISM CORRECTION

cat("\n=== Starting Bootstrap Optimism Correction ===\n")
tuning_result                                                    <-
at_xgboost_6$tuning_instance$result_learner_param_vals
set.seed(123)
n_boot <- 500
boot_results <- data.frame(
  Iteration = 1:n_boot,
  Apparent_AUROC = numeric(n_boot),
```

```
  Original_AUROC = numeric(n_boot),
  Optimism_AUROC = numeric(n_boot)
)
train_data <- train_processed_6$data()
for (i in 1:n_boot) {
  if (i %% 50 == 0) cat("Bootstrap iteration", i, "of", n_boot, "\n")
  boot_indices <- sample(1:nrow(train_data), size = nrow(train_data),
replace = TRUE)
  boot_df <- train_data[boot_indices, ]
  boot_task <- as_task_classif(x = boot_df, target = "Nephrotoxicity",
positive = "1")
  boot_learner <- lrn("classif.xgboost",
                      nrounds = tuning_result$nrounds,
                      eta = final_params$eta,
                      max_depth = final_params$max_depth,
                      min_child_weight =
final_params$min_child_weight,
                      subsample = final_params$subsample,
                      colsample_bytree =
final_params$colsample_bytree,
                      lambda = final_params$lambda,
                      alpha = final_params$alpha,
                      gamma = final_params$gamma,
                      predict_type = "prob"
  )

  boot_learner$train(boot_task)
  boot_pred_apparent <- boot_learner$predict(boot_task)
  apparent_auc <- boot_pred_apparent$score(msr("classif.auc"))
  boot_pred_original <- boot_learner$predict(train_processed_6)
  original_auc <- boot_pred_original$score(msr("classif.auc"))
  boot_results$Apparent_AUROC[i] <- apparent_auc
  boot_results$Original_AUROC[i] <- original_auc
  boot_results$Optimism_AUROC[i] <- apparent_auc - original_auc
}
mean_optimism_auc <- mean(boot_results$Optimism_AUROC)
sd_optimism_auc <- sd(boot_results$Optimism_AUROC)
optimism_corrected_auc <- main_test_auc - mean_optimism_auc
boot_corrected_values <- main_test_auc -
boot_results$Optimism_AUROC
boot_ci_auc_lower <- quantile(boot_corrected_values, 0.025)
boot_ci_auc_upper <- quantile(boot_corrected_values, 0.975)
```

```
cat("\n=== Bootstrap Results ===\n")
cat(sprintf("    Apparent AUROC (mean ± SD): %.3f ± %.3f\n",
                  mean(boot_results$Apparent_AUROC),
sd(boot_results$Apparent_AUROC)))
cat(sprintf("    Original AUROC (mean ± SD): %.3f ± %.3f\n",
                  mean(boot_results$Original_AUROC),
sd(boot_results$Original_AUROC)))
cat(sprintf("    Mean optimism: %.3f ± %.3f\n", mean_optimism_auc,
sd_optimism_auc))
cat(sprintf("    Main Test AUROC: %.3f\n", main_test_auc))
cat(sprintf("    Optimism-Corrected AUROC: %.3f\n",
optimism_corrected_auc))
cat(sprintf("    95%% CI: %.3f - %.3f\n", boot_ci_auc_lower,
boot_ci_auc_upper))

# Table S11
table_s11 <- data.frame(
  Metric = c("Apparent AUROC (mean ± SD)",
             "Original Training AUROC (mean ± SD)",
             "Optimism AUROC (mean ± SD)",
             "Main Test AUROC (6-feature)",
             "Optimism-Corrected AUROC",
             "95% CI Lower",
             "95% CI Upper"),
  Value = c(
    sprintf("%.3f ± %.3f", mean(boot_results$Apparent_AUROC),
sd(boot_results$Apparent_AUROC)),
    sprintf("%.3f ± %.3f", mean(boot_results$Original_AUROC),
sd(boot_results$Original_AUROC)),
    sprintf("%.3f ± %.3f", mean_optimism_auc, sd_optimism_auc),
    sprintf("%.3f", main_test_auc),
    sprintf("%.3f", optimism_corrected_auc),
    sprintf("%.3f", boot_ci_auc_lower),
    sprintf("%.3f", boot_ci_auc_upper)
  )
)

write.csv(table_s11,    "Table_S11_Bootstrap_Results_6feature.csv",
row.names = FALSE)
figure_s10_data <- data.frame(
  AUROC          =          c(boot_results$Apparent_AUROC,
boot_results$Original_AUROC),
```

```
  Type = rep(c("Apparent AUROC (bootstrap sample)",
                "Original AUROC (full training set)"), each = n_boot)
)
apparent_density <- density(boot_results$Apparent_AUROC)
original_density <- density(boot_results$Original_AUROC)

figure_s10 <- ggplot(figure_s10_data, aes(x = AUROC, fill = Type)) +
  geom_density(alpha = 0.5, linewidth = 0.8) +
  geom_vline(xintercept = main_test_auc, linetype = "dashed",
             color = "red", linewidth = 1) +
  annotate("text", x = main_test_auc + 0.005,
           y = max(apparent_density$y) * 0.85,
           label = paste("6-feature test =", round(main_test_auc, 3)),
           color = "red", size = 4, fontface = "bold", hjust = 0) +
  geom_vline(xintercept = optimism_corrected_auc, linetype =
"dotted",
             color = "blue", linewidth = 1) +
  annotate("text", x = optimism_corrected_auc - 0.005,
           y = max(apparent_density$y) * 0.65,
           label = paste("Corrected =",
round(optimism_corrected_auc, 3)),
           color = "blue", size = 4, fontface = "bold", hjust = 1) +
  annotate("segment", x = mean(boot_results$Original_AUROC),
           xend = mean(boot_results$Apparent_AUROC),
           y = max(apparent_density$y) * 0.4,
           yend = max(apparent_density$y) * 0.4,
           arrow = arrow(ends = "both", length = unit(0.2, "cm")),
           color = "darkgreen", linewidth = 1) +
  annotate("text", x = (mean(boot_results$Original_AUROC) +
mean(boot_results$Apparent_AUROC)) / 2,
           y = max(apparent_density$y) * 0.45,
           label = paste("Optimism =", round(mean_optimism_auc,
3)),
           color = "darkgreen", size = 3.5, fontface = "bold") +
  scale_x_continuous(limits = c(0.75, 1.0), breaks = seq(0.75, 1.0,
0.05),
                     oob = scales::squish) +
  scale_fill_manual(values = c("Apparent AUROC (bootstrap sample)"
= "#E69F00",
                               "Original AUROC (full training
set)" = "#56B4E9")) +
  labs(x = "AUROC", y = "Density",
```

```
        title = "Figure S8. Bootstrap Optimism Correction (6-Feature Model)",
        subtitle = paste("Mean optimism =", sprintf("%.3f", mean_optimism_auc),
                         "| Corrected AUROC =", sprintf("%.3f", optimism_corrected_auc),
                         "(95% CI:", sprintf("%.3f-%.3f", boot_ci_auc_lower, boot_ci_auc_upper, ")"))) +
  theme_minimal(base_size = 12) +
  theme(
    plot.title = element_text(size = 14, face = "bold"),
    plot.subtitle = element_text(size = 11, color = "gray40"),
    legend.position = "top",
    legend.title = element_blank(),
    panel.border = element_rect(color = "black", fill = NA, linewidth = 0.5)
  )

ggsave("Figure_S8_Bootstrap_Optimism_6feature.png", figure_s10,
        width = 12, height = 7, dpi = 300)
cat("\nFigure S8 saved successfully\n")
print(figure_s8)

## Step 9: Feature Importance and SHAP

```{r}
set.seed(123)
df <- read.xlsx("Rawdata.xlsx") %>%
  select(-id,-BNP) %>%
  mutate(across(where(is.character), as.factor))

df$Nephrotoxicity <- factor(df$Nephrotoxicity, levels = c("no", "yes"))

task <- as_task_classif(df, target = "Nephrotoxicity", positive = "yes")

preprocess_pipe <- po("imputehist") %>>%
  po("imputemode") %>>%
  po("scale", affect_columns = selector_type("numeric")) %>>%
  po("encode", method = "treatment", affect_columns = selector_type("factor"))
processed_task <- preprocess_pipe$train(task)[[1]]
```
```

```
lrn_xgboost <- lrn("classif.xgboost",
                                predict_type = "prob",
                                nrounds = 100,
                                eta = 0.1,
                                max_depth = 6,
                                verbose = 0
)

lrn_xgboost$train(processed_task)


processed_data <- processed_task$data()

feature_names <- setdiff(colnames(processed_data), "Nephrotoxicity")
X_all <- processed_data[, feature_names, with = FALSE]
set.seed(456)
sample_idx <- sample(nrow(X_all), min(1000, nrow(X_all)))
X_explain <- X_all[sample_idx, , drop = FALSE]

pred_fun_xgb <- function(model, newdata) {
   newdata <- as.data.frame(newdata)

   newdata_matrix <- as.matrix(newdata)


   pred <- predict(model, newdata_matrix)


   if (is.matrix(pred) && ncol(pred) == 2) {
      return(pred[, 2])
   } else {
      return(pred)
   }
}


bg_X <- X_all[sample(nrow(X_all), 1452), , drop = FALSE]
ps <- kernelshap(
   object = lrn_xgboost$model,
   X = X_explain,
   bg_X = bg_X,
   pred_fun = pred_fun_xgb
```

```
)

sample_labels <- processed_data$Nephrotoxicity[sample_idx]
table(sample_labels)
neg_idx <- which(sample_labels == "no")[1]
pos_idx <- which(sample_labels == "yes")[1]
sv <- shapviz(ps)

p_bee <- sv_importance(sv, "bee") +
  theme_bw() +
  theme(
    axis.title = element_text(size = 14, face = "bold"),
    axis.text = element_text(size = 12, face = "bold"),
    legend.title = element_text(size = 12, face = "bold"),
    legend.text = element_text(size = 11, face = "bold"),
    plot.title = element_text(size = 16, face = "bold", hjust = 0.5),
    panel.background = element_rect(fill = "white"),
    plot.background = element_rect(fill = "white"),
    panel.grid = element_blank()
  )

ggsave("SHAP_Beeswarm_XGB.png", p_bee,
       dpi = 600, width = 14, height = 10,
       device = "png", bg = "white")
p_force_neg <- sv_force(sv, row_id = neg_idx, fill_colors = c("blue",
"red")) +
  ggtitle("Force Plot - (No Nephrotoxicity)") +
  theme(plot.title = element_text(size = 12, face = "bold", hjust = 0.5))
p_force_neg
p_force_pos <- sv_force(sv, row_id = pos_idx, fill_colors = c("blue",
"red")) +
  ggtitle("Force Plot - (Nephrotoxicity)") +
  theme(plot.title = element_text(size = 12, face = "bold", hjust = 0.5))
p_force_pos
p_combined <- p_force_neg / p_force_pos+
  plot_annotation(tag_levels = "A")

ggsave("forceplot_XGB.png", p_combined,
       dpi = 600, width = 10, height = 8,
       device = "png", bg = "white")
p_waterfall_neg <- sv_waterfall(sv, row_id = neg_idx, max_display = 8,
                                fill_colors = c("#3498db",
```

```
"#e74c3c")) +
  ggtitle("SHAP Waterfall Plot - XGBoost") +
  theme_bw() +
  theme(
    panel.background = element_rect(fill = "white"),
    plot.background = element_rect(fill = "white"),
    text = element_text(face = "bold", size = 12)
  )
p_waterfall_neg
p_waterfall_pos <- sv_waterfall(sv, row_id = pos_idx, max_display = 8,
                                fill_colors = c("#3498db",
"#e74c3c")) +
  ggtitle("SHAP Waterfall Plot - XGBoost") +
  theme_bw() +
  theme(
    panel.background = element_rect(fill = "white"),
    plot.background = element_rect(fill = "white"),
    text = element_text(face = "bold", size = 12)
  )
p_waterfall_pos
p_combined_waterfall <- p_waterfall_neg / p_waterfall_pos+
  plot_annotation(tag_levels = "A")
ggsave("SHAP_Waterfall_XGB.png", p_combined_waterfall,
       dpi = 600, width = 10, height = 8,
       device = "png", bg = "white")
set.seed(123)
n_bootstrap <- 100
imp_list <- list()

for (i in 1:n_bootstrap) {

  boot_idx <- sample(processed_task$nrow, replace = TRUE)
  boot_task <- processed_task$clone()$filter(boot_idx)
  lrn_temp <- lrn("classif.xgboost",
                  predict_type = "prob",
                  nrounds =
lrn_xgboost$param_set$values$nrounds,
                  eta = lrn_xgboost$param_set$values$eta,
                  max_depth =
lrn_xgboost$param_set$values$max_depth,
                  verbose = 0
  )
```

```
  lrn_temp$train(boot_task)
  imp <- lrn_temp$importance()
  imp_df_temp <- data.frame(
    Feature = names(imp),
    Importance = as.numeric(imp),
    boot_id = i
  )
  imp_list[[i]] <- imp_df_temp
}
imp_all <- bind_rows(imp_list)
imp_summary <- imp_all %>%
  group_by(Feature) %>%
  summarise(
    Mean = mean(Importance),
    SD = sd(Importance),
    SE = SD / sqrt(n()),
    CI_lower = quantile(Importance, 0.025),
    CI_upper = quantile(Importance, 0.975),
    .groups = 'drop'
  ) %>%
  arrange(desc(Mean))
p_importance <- ggplot(imp_summary, aes(x = reorder(Feature, Mean),
y = Mean)) +
  geom_col(fill = "steelblue", alpha = 0.7) +
  geom_errorbar(aes(ymin = CI_lower, ymax = CI_upper),
                width = 0.3, color = "black", linewidth = 0.8) +
  geom_text(aes(label = sprintf("%.3f", Mean)),
            hjust = -0.2, size = 3.5, fontface = "bold", color =
"darkred") +
  coord_flip() +
  expand_limits(y = max(imp_summary$CI_upper) * 1.15) +
  ggtitle("XGBoost Feature Importance (Gain)\nwith 95% Bootstrap
CI") +
  xlab("Feature") +
  ylab("Importance (Mean ± 95% CI)") +
  theme_bw() +
  theme(
    panel.background = element_rect(fill = "white"),
    plot.background = element_rect(fill = "white"),
    legend.position = "none",
    axis.text = element_text(size = 12, face = "bold"),
    axis.title = element_text(size = 14, face = "bold"),
```

```
    plot.title = element_text(size = 16, face = "bold", hjust = 0.5)
  )

ggsave("XGBoost_Importance_with_CI.png", p, dpi = 600, width = 14, height = 10, bg = "white")
```

Data Availability
The data that support the findings of this study are available from the corresponding author upon reasonable request. The data are not publicly available due to privacy and ethical restrictions that protect the confidentiality of patient information from electronic health records.

Code Availability

The complete analytical code is provided in this document. For reproducibility, the code requires:
- R version 4.4.2 or higher
- The mlr3 ecosystem (mlr3, mlr3pipelines, mlr3tuning, mlr3learners, mlr3measures)
- Access to the study dataset (available upon reasonable request)

End of Supplementary Code