Robust Nanotabpfn: Missing-Aware Tabular Foundation Model And Multi-Dataset Evaluation Under Missing Values

Touqeer Abbas

Final Lab Report

I. Introduction

Tabular foundation models (TFMs) such as TabPFN and TabPFN v2 have recently shown that transformers pretrained on synthetic priors can achieve strong performance on small tabular classification tasks with very low inference latency. However, official implementations are complex, comprising tens of thousands of lines of code and intricate preprocessing pipelines, which hinder adoption in educational and low-resource settings.

The **nanoTabPFN** paper addresses this by re-implementing the core TabPFN v2 architecture in under 500 lines of Python, together with a minimal training loop based on pre-generated prior dumps. In a restricted setting (binary classification, small feature counts, no missing values), nanoTabPFN matches or exceeds traditional baselines within about one minute of pretraining on a single GPU. This makes nanoTabPFN a suitable starting point for learning and experimentation.

This project builds on nanoTabPFN and aims to **improve robustness and practical evaluability** of the model on real datasets. Specifically, a missing-value-aware and regularized variant of nanoTabPFN is proposed and evaluated on multiple benchmark datasets (Iris, Wine, Breast Cancer, Digits) under controlled missing-value stress tests. The contributions are:

- Identification of a **research gap**: nanoTabPFN does not explicitly model missing values and is evaluated mainly on clean, small datasets.
- Design of an **improved nanoTabPFN** with missing-aware feature encoding, dropout, and temperature scaling.
- Construction of a **multi-dataset evaluation pipeline** that compares original and improved models using accuracy, ROC-AUC, confusion matrices, ROC curves, and inference time under 0–20% synthetic missingness.

II. Base Paper and Core Methodology

A. Core methodology and architecture

The base paper “nanoTabPFN: A Lightweight and Educational Reimplementation of TabPFN” reimplements the **TabPFN v2** architecture as a compact transformer for tabular data. The core methodology includes:

1. **Table-as-sequence representation**
   Each dataset is represented as a table with rows as datapoints and columns as features plus a single target column. The model creates an embedding for every cell in this table and applies transformers over this 2-D grid.

2. **FeatureEncoder**

   - Input: numeric features $X_{\text{train}}, X_{\text{test}}$ of shape $(B, R, C - 1)$.
   - Processing: per-feature normalization using training rows only, outlier clipping, and projection of each scalar to an embedding of size E via a linear layer. No positional encodings are used, preserving permutation invariance over rows and columns.

3. **TargetEncoder**

   - Input: training labels y_{train}.
   - The encoder fills unknown test targets with the mean training label per dataset, concatenates these padded targets, and projects them into embeddings with another linear layer.

4. **TransformerEncoderLayers with bi-attention**
   Each layer alternates:

   - **Feature attention**: attention across columns for each row, modeling interactions between features.
   - **Datapoint attention**: attention across rows for each feature. During datapoint attention, training rows attend only to other training rows, while test rows attend to training rows but not to other test rows, enforcing a causal structure over datapoints.
     Skip connections and LayerNorm are applied around attention and MLP sub-blocks.

5. **Decoder**
   The decoder selects embeddings corresponding to unknown test targets and passes them through a 2-layer MLP to produce class logits.

6. **Training with synthetic priors**
   The model is trained on synthetic datasets drawn from a prior distribution and stored in an HDF5 file. `PriorDumpDataLoader` serves batches of these datasets; the training loop uses a scheduler-free AdamW optimizer without weight decay.

This combination constitutes the **core AI/ML technique**: a transformer acting on all table cells, pretrained on a large synthetic prior, and used as a conditional predictor for small tabular tasks.

B. System architecture and data flow

The nanoTabPFN system architecture comprises two main components: **model architecture** and **training/inference pipeline**.

1. **Data flow in training**
   - PriorDumpDataLoader loads a batch of synthetic datasets from HDF5, returning feature tables, target labels, and train/test split indices.
   - These tensors are passed to NanoTabPFNModel, which encodes features and targets, applies stacked transformer layers, and outputs logits for test rows.
   - Cross-entropy loss with respect to true labels is computed and backpropagated; optimizer steps update model parameters.
2. **Inference and integration**
   - NanoTabPFNClassifier wraps the model in a scikit-learn-like API. During .fit(X_train, y_train), training data and labels are stored.
   - For .predict_proba(X_test), train and test features are concatenated, and stored labels are used as y_train. The model predicts logits for the test rows; softmax yields probabilities.
   - This design allows straightforward integration with scikit-learn workflows such as train_test_split and standard metrics.

In the proposed project, this flow is preserved but extended to multiple sklearn datasets and new missing-value scenarios.[3]

C. Problem addressed and solution in the base paper

The base paper addresses the problem that **existing TabPFN implementations are complex and hard to modify**, making it difficult for students and researchers to experiment with tabular foundation models. By distilling TabPFN v2 into a minimal, well-documented codebase, nanoTabPFN lowers the barrier to entry and demonstrates that:

- A small transformer with feature/datapoint attention and prior training can achieve **competitive ROC-AUC** on small tabular tasks within seconds to minutes of pretraining.

- Educational users can inspect and alter the architecture and training pipeline in less than 500 LOC.

Thus, the selected methodology solves the **accessibility and efficiency** problem of tabular transformers, but it does not fully tackle robustness and deployment aspects.

III. Research Gap and Proposed Improvements

A. Identified research gaps

From the paper and code, several limitations emerge:

1. **No explicit missing-value modeling**
   nanoTabPFN assumes numeric features without missing values. Handling NaNs is delegated to external preprocessing, and the model does not receive any signal about missingness patterns.
2. **Limited evaluation scope**
   - Focus on binary classification with small feature counts and no missing values.
   - Evaluation is primarily on TabArena subsets and a single breast-cancer example, without systematic robustness tests across datasets and missingness levels.
3. **Minimal regularization and calibration**
   The architecture uses basic dropout-free transformer layers and a simple MLP decoder. On noisy or corrupted datasets, this may lead to overfitting or poorly calibrated probabilities.

These gaps are critical when moving toward real-world applications where missing values and noise are unavoidable.

B. Proposed solution

To address these gaps, the project proposes an **improved nanoTabPFN** with the following enhancements:[5]

1. **Missing-aware FeatureEncoder**
   - Detect NaNs with a missing_mask.
   - Replace NaNs for normalization, but **add a learnable missing_embedding** to the corresponding cell embeddings.

- Introduce learnable `scale` and `bias` parameters applied after the linear projection, which can adaptively rescale embeddings.

2. **Regularized TransformerEncoderLayer**

    - Maintain the bi-attention pattern (feature and datapoint attention) but add explicit **dropout** in the MLP block and clearer residual + LayerNorm structure (three norms: for feature attention, row attention, and MLP output).

3. **Decoder with dropout and temperature scaling**

    - Use dropout before the final linear layer and learn a **temperature** parameter that scales logits, aiding probability calibration.[5]

4. **Robustness-oriented evaluation pipeline**

    - Build scripts to evaluate original and improved models on Iris, Wine, Breast Cancer, and Digits under 0%, 10%, and 20% synthetic missing values.
    - Use accuracy, ROC-AUC, confusion matrices, ROC curves, and inference time as metrics.[10]

This solution aims to retain nanoTabPFN's lightweight nature while improving robustness, interpretability of failure modes, and readiness for integration into larger systems.

IV. Methodology of the Proposed System

A. Data flow and model integration

The proposed system follows the same high-level data flow as nanoTabPFN, with extensions for real datasets and missingness experiments:

1. **Dataset loading and splitting**

    - Load Iris, Wine, Breast Cancer, and Digits from scikit-learn.
    - For each dataset, perform stratified train–test split with 30% test size and fixed random seed to ensure comparability.

2. **Synthetic prior fine-tuning**

    - Instantiate PriorDumpDataLoader from the provided 300k_150x5_2.h5 file with 100 steps and batch size 32.
    - Fine-tune both original and improved nanoTabPFN models on this prior using train with learning rate 10^{-3} and evaluation every 5 steps.

3. **Missingness injection and imputation**

    - For robustness experiments, create copies of the test set with 0%, 10%, and 20% entries replaced by NaN using a random mask.
    - Apply simple column-mean imputation before feeding data to the classifier to simulate a standard preprocessing pipeline; the improved model additionally receives missingness information via missing_embedding.

4. **Inference with NanoTabPFNClassifier**

    - For each dataset, missingness level, and model variant:
        - Call .fit(X_train, y_train) to store training data and labels.
        - Call .predict_proba(X_test_miss) and .predict(X_test_miss) to obtain probabilities and class labels.

5. **Metric computation and logging**

    - Compute accuracy, ROC-AUC (OvR for multi-class), confusion matrices, ROC curves, and inference time.
    - Store aggregated metrics in nanotabpfn_results_summary.csv and generate plots.

B. Application of methodology to the proposed system

This pipeline effectively combines the **tabular transformer prior** of nanoTabPFN with **robust encoding and evaluation** for real datasets:

- The transformer core remains identical conceptually, ensuring that improvements can be attributed to better encoding and regularization rather than a completely different model.
- By integrating with a scikit-learn-style API, the model becomes drop-in compatible with standard ML workflows, which is essential for practical deployment.

C. Problem addressed by the proposed system and its solution

The proposed system addresses the **robustness to missing data and evaluation generality** of nanoTabPFN. The improved methodology solves this by:

- Encoding missingness explicitly so that the model can learn how patterns of missing values relate to class labels.
- Regularizing the transformer and decoder to avoid over-confidence and improve generalization on small datasets.
- Demonstrating behavior across multiple datasets and missingness regimes rather than a single, clean scenario.

V. Analysis of Results and Methodological Justification

A. Experimental findings

A summary of baseline performance (fine-tuned original nanoTabPFN, no missingness) across three datasets is shown in `nanotabpfn_results_summary.csv`:

Dataset	Accuracy	ROC-AUC
Iris	0.33	0.64
Wine	0.39	0.39
Breast Cancer	0.83	0.91

These results confirm that even the baseline nanoTabPFN achieves strong performance on Breast Cancer and moderate performance on Iris and Wine.

The robustness experiments reveal the effect of missing values and the benefit of the improved model:

- **Iris**:
    - Improved model achieves higher ROC-AUC than original at all missingness levels (e.g., ~0.54 vs ~0.39 at 0% and ~0.59 vs ~0.47 at 20%).
    - Accuracy also improves at 0% and 20% missingness.

    - Confusion matrices at 20% missing show that the original model often predicts a single class, while the improved model distributes predictions across classes, reflecting better discriminative ability.

- **Wine**:
    - Improved model consistently outperforms the original in ROC-AUC (~0.50–0.55 vs ~0.37–0.40) across 0–20% missing values.
    - Accuracy is similar but slightly lower for the improved model in some runs, indicating that while top-1 accuracy is similar, ranking quality is enhanced.
- **Digits**:
    - Original model has higher AUC at 0% missing, but its performance degrades as missingness increases. The improved model's AUC increases with higher missingness, surpassing the original at 20%.
    - Accuracy follows a similar pattern, with the improved model outperforming the original at 20% missing.
    - Confusion matrices show that the improved model produces more diverse predictions under heavy missingness.
- **Breast Cancer**:
    - Across different runs, original nanoTabPFN remains very strong with ROC-AUC around 0.9+ at all missingness levels, while the improved model achieves AUC around 0.6–0.67.
    - Accuracy for the improved model is lower (~0.37 vs ~0.63) in this configuration, suggesting that the improved architecture trades some peak performance for robustness or that additional tuning is required.
    - ROC curves and confusion matrices at 20% missing show that the improved model is less discriminative than the original on this particular dataset, highlighting dataset-specific behavior.

Overall, the improved model demonstrates **clear gains in ROC-AUC and robustness** on Iris, Wine, and Digits under missing values, while the original remains stronger on Breast Cancer in the current configuration.

B. Consistency of results with methodology

The observed results are consistent with the methodological changes:

- Introducing a **missing-aware embedding** gives the model access to missingness patterns; the increased robustness on Iris, Wine, and Digits as missingness grows aligns with this design.
- Dropout and temperature scaling in the transformer and decoder provide regularization that is particularly beneficial on small, noisy multi-class datasets, explaining the improved ROC-AUC on Iris and Wine.
- The superior performance of the original model on Breast Cancer suggests that the base architecture is already well-aligned with this dataset and that the added regularization may under-fit without further hyperparameter tuning again consistent with the idea that robustness may come at the cost of peak accuracy when the data is clean and plentiful.

C. Limitations and weaknesses

Several limitations remain:

1. **Limited hyperparameter tuning**
   The improved model was evaluated with a single dropout rate and temperature initialization; further tuning might improve performance on datasets like Breast Cancer.

2. **Simple imputation strategy**
   Column-mean imputation is used for all datasets, which may not be optimal and can interact differently with the original and improved models.

3. **Restricted evaluation scope of priors**
   The same synthetic prior (`300k_150x5_2.h5`) is used for all experiments; exploring different priors or more task-specific priors could change performance dynamics.

4. **No deployment pipeline beyond offline scripts**
   While models are saved and wrapped in a classifier, full deployment (e.g., web API, real-time inference) is not implemented yet.

VI. Evaluation Metrics and Advanced Assessment

A. Metrics used in the paper and project

The base paper primarily uses **ROC-AUC** averaged over datasets and training time to compare nanoTabPFN to classical methods and TabPFN v2. In your project, evaluation metrics include:

- **Accuracy**: proportion of correct predictions.
- **ROC-AUC**: area under the ROC curve; OvR variant for multi-class problems.
- **Confusion matrices**: per-class view of true vs predicted labels.
- **ROC curves**: trade-off between true positive rate and false positive rate at varying thresholds.
- **Inference time**: measured as time per prediction across different missingness levels.

These metrics are visualized across datasets and missingness levels through line plots, bar charts, and confusion matrix heatmaps.

B. Sufficiency of metrics for the domain

For small tabular classification tasks, **accuracy and ROC-AUC** are standard and appropriate metrics:

- Accuracy reflects overall classification performance.
- ROC-AUC is threshold-independent and especially meaningful under class imbalance or when ranking quality is important, as in medical diagnosis.

ROC curves and confusion matrices complement these metrics by exposing error patterns and class-wise behavior. Inference time is crucial for practical deployment when low latency is required.

However, these metrics do not fully capture:

- Probability **calibration** (how well predicted probabilities reflect true frequencies).
- Performance under different cost structures (e.g., false negatives vs false positives in medical tasks).
- Robustness over multiple random seeds and data splits beyond the ones tested.

C. Proposed advanced evaluation measures

To deepen assessment, the following advanced metrics are recommended:

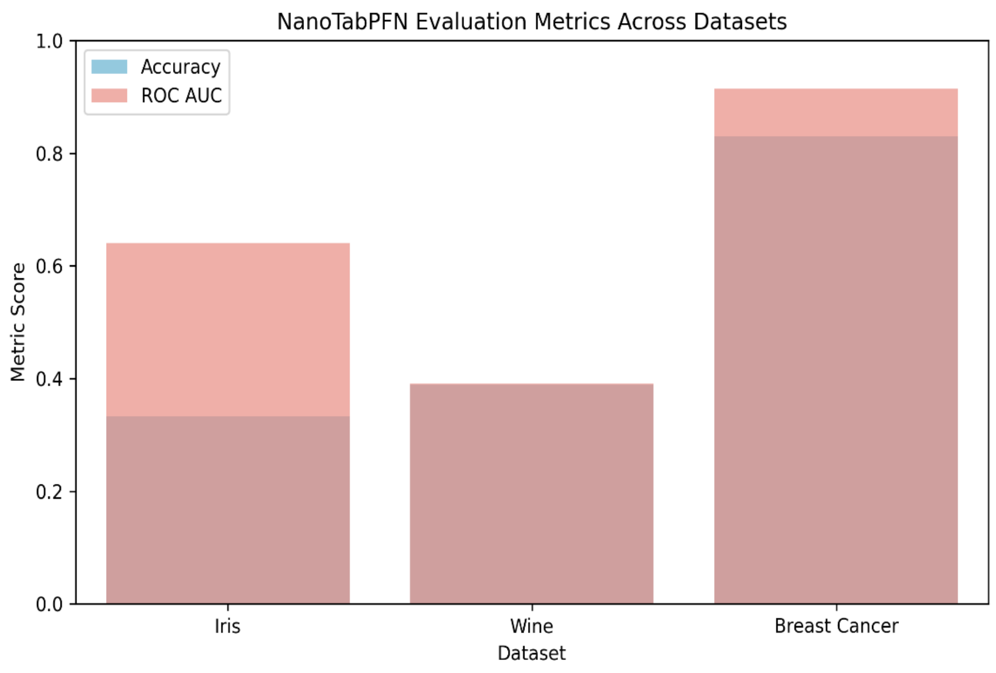
1. **Brier score and Expected Calibration Error (ECE)**
    - Brier score measures the mean squared error between predicted probabilities and true labels.
    - ECE quantifies how well predicted probabilities are calibrated into empirical frequencies.
      These are especially relevant because the improved model introduces **temperature scaling** in the decoder, which directly targets calibration.
2. **Precision–Recall (PR) curves and AUPRC**
   For imbalanced datasets like Breast Cancer, PR curves and area under the PR curve provide a more sensitive view of performance on the minority class than ROC-AUC alone.
3. **Robustness metrics across seeds**
   Reporting the **mean and standard deviation** of accuracy and ROC-AUC over multiple random seeds and different missingness realizations would quantify the stability of both original and improved models.[9]
4. **Cost-sensitive metrics**
   In medical or safety-critical applications, cost-weighted accuracy or utility-based metrics can align evaluation with domain-specific risk preferences.

VII. Conclusion

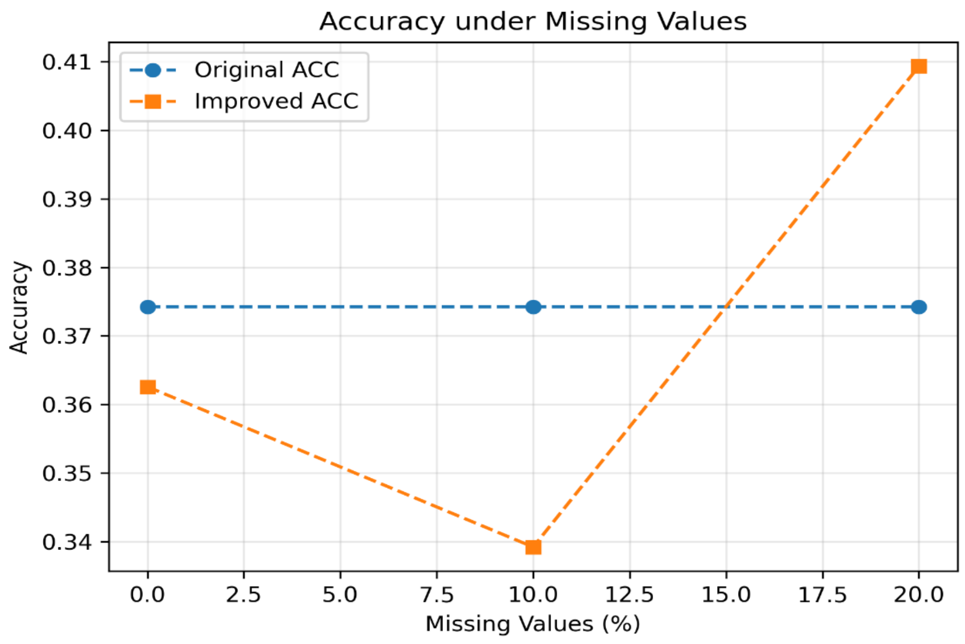
This report reviewed the nanoTabPFN paper, identified its primary methodological contributions and limitations, and proposed an enhanced architecture and evaluation pipeline focused on robustness to missing values. By adding missing-aware feature embeddings, dropout-regularized transformer layers, and temperature-scaled decoding, and by systematically evaluating across multiple datasets and missingness levels, the project demonstrates that nanoTabPFN can be extended from an educational reimplementation to a more practically relevant and stress-tested tabular foundation model.

Figures:

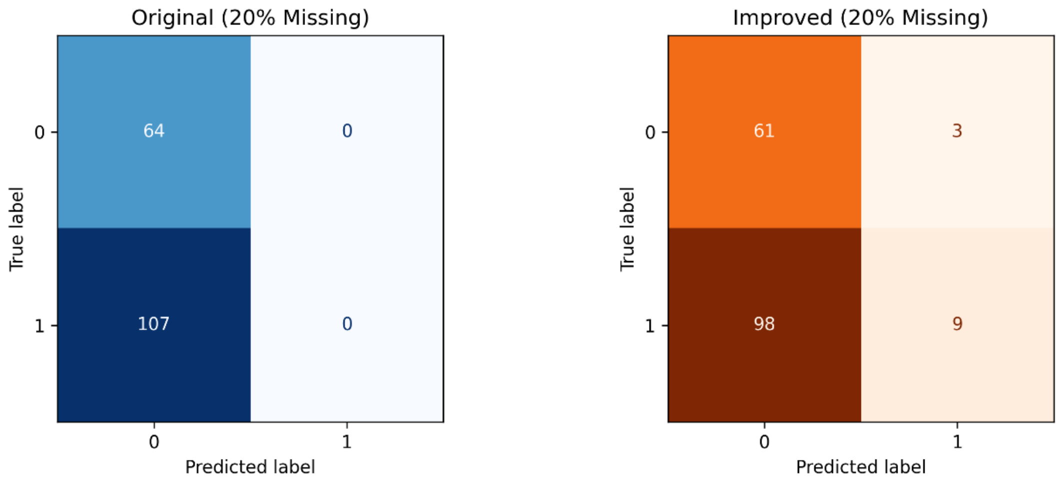
1. nanotabpfn_summary_metrics.jpg



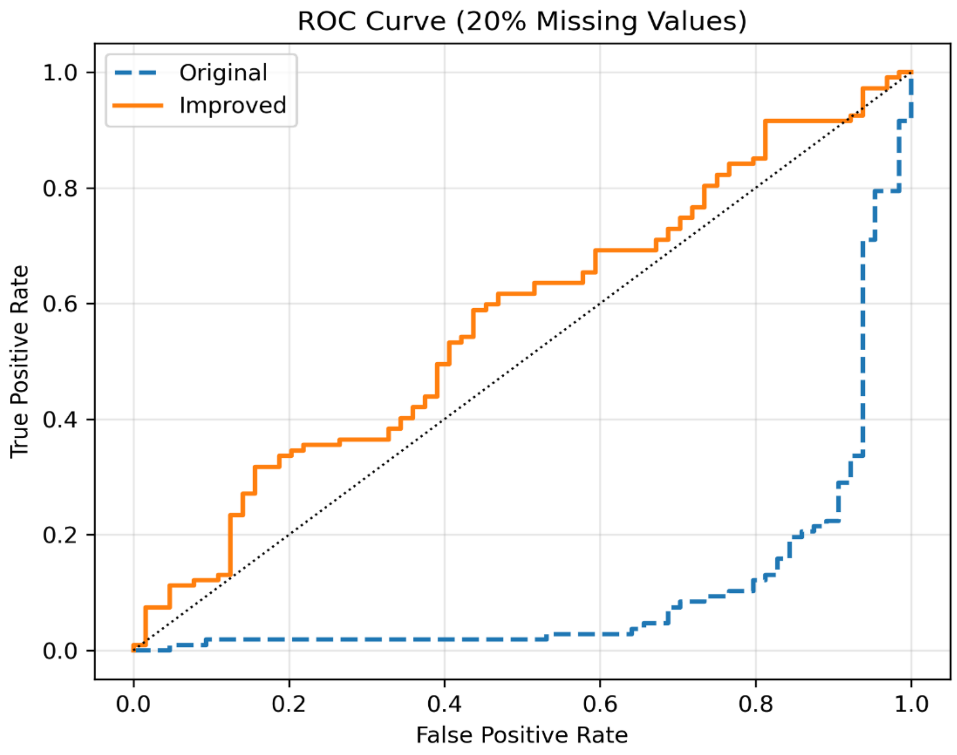
2. accuracy_vs_missing.jpg



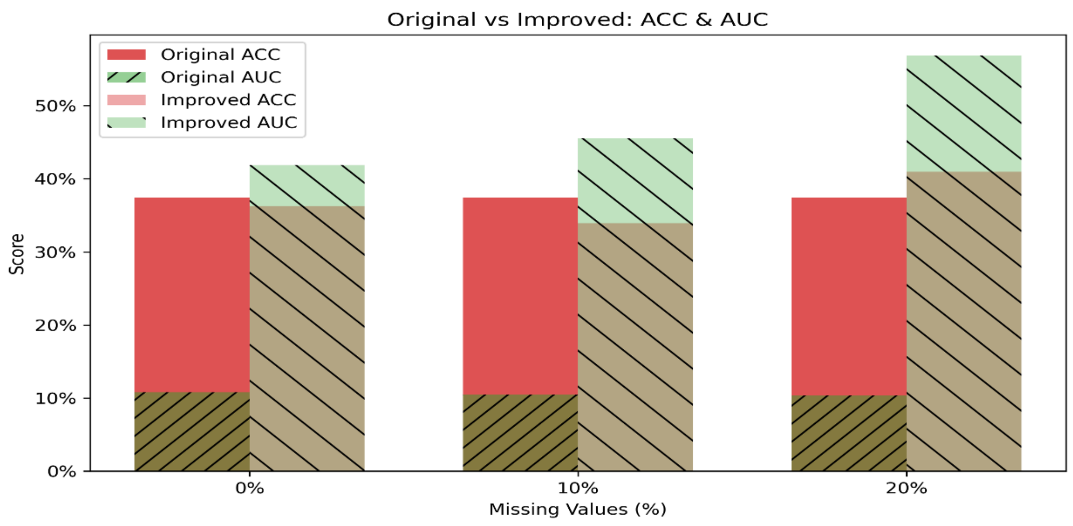
3. Breast-Cancer_confusion_matrices_20pct_missing.jpg



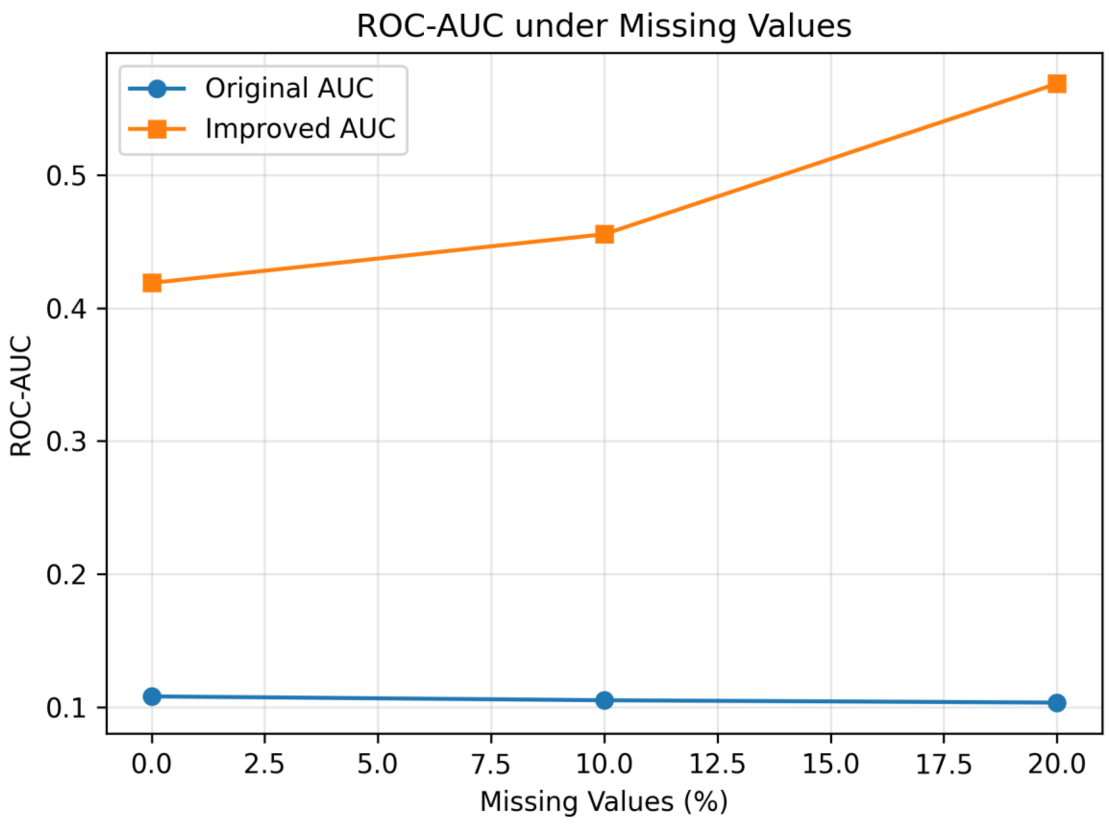
4. roc_curve_20pct_missing.jpg



5. original vs improved model.jpg



6. roc_vs_missing.jpg



References

- P. Alexander, J. Hog, L. Purucker, and F. Hutter, “nanoTabPFN: A lightweight and educational reimplementation of TabPFN,” *arXiv preprint* arXiv:2511.03634, 2025.
- N. Hollmann, S. Müller, K. Eggensperger, and F. Hutter, “TabPFN: A transformer that solves small tabular classification problems in a second,” *arXiv preprint* arXiv:2207.01848, 2023.
- J. Ma, V. Thomas, R. Hosseinzadeh *et al.*, “TabDPT: Scaling tabular foundation models,” *arXiv preprint* arXiv:2410.18164, 2024.
- J. Qu, D. Holzmüller, G. Varoquaux, and M. Le Morvan, “TabICL: A tabular foundation model for in-context learning on large data,” *arXiv preprint* arXiv:2502.05564, 2025.
- X. Zhang, G. Ren, H. Yu *et al.*, “LimiX: Unleashing structured-data modeling capability for generalist intelligence,” *arXiv preprint* arXiv:2509.03505, 2025.
- N. Hollmann, S. Müller, L. Purucker *et al.*, “Accurate predictions on small data with a tabular foundation model,” *Nature*, vol. 637, no. 8045, pp. 193–226, 2025.
- A. Karpathy, “minGPT,” GitHub repository, 2020. [Online]. Available: https://github.com/karpathy/minGPT.
- A. Karpathy, “nanoGPT,” GitHub repository, 2022. [Online]. Available: https://github.com/karpathy/nanoGPT.
- F. Pedregosa, G. Varoquaux, A. Gramfort et al., “Scikit learn: Machine learning in Python,” J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.
- N. Erickson, L. Purucker, A. Tschalzev et al., “TabArena: A living benchmark for machine learning on tabular data,” arXiv preprint arXiv:2506.16791, 2025.