TabImpute: Accurate and Fast Zero-Shot Missing-Data Imputation with a Pre-Trained Transformer

Jacob Feitelberg
Industrial Engineering & Operations Research
Columbia University
jef2182@columbia.edu

Dwaipayan Saha
Industrial Engineering & Operations Research
Columbia University
ds4386@columbia.edu

Kyuseong Choi
Statistics and Data Science
Cornell Tech
kc728@cornell.edu

Zaid Ahmad
Department of Statistics
Columbia University
za2364@columbia.edu

Anish Agarwal
Industrial Engineering & Operations Research
Columbia University
aa5194@columbia.edu

Raaz Dwivedi
Operations Research & Information Engineering
Cornell Tech
rd597@cornell.edu

Abstract

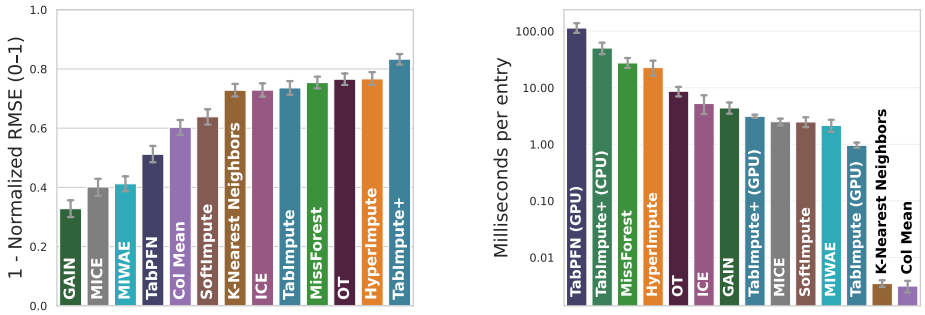
Missing data is a pervasive problem in tabular settings. Existing solutions range from simple averaging to complex generative adversarial networks. However, due to huge variance in performance across real-world domains and time-consuming hyperparameter tuning, no default imputation method exists. Building on TabPFN, a recent tabular foundation model for supervised learning, we propose TabImpute, a pre-trained transformer that delivers accurate and fast zero-shot imputations requiring no fitting or hyperparameter tuning at inference-time. To train and evaluate TabImpute, we introduce (i) an entry-wise featurization for tabular settings, which enables a $100\times$ speedup over the previous TabPFN imputation method, (ii) a synthetic training data generation pipeline incorporating realistic missingness patterns, which boosts test-time performance, and (iii) MissBench, a comprehensive benchmark for evaluation of imputation methods with 42 OpenML datasets and 13 missingness patterns. MissBench spans domains such as medicine, finance, and engineering, showcasing TabImpute's robust performance compared to 11 established imputation methods.

1 Introduction

Missing data is ubiquitous across tabular datasets, affecting statisticians, economists, health officials, and businesses. For example, healthcare datasets may lack some recorded blood pressure measurements, or datasets merged from multiple sources may only share partial features. Regardless of the

AI for Tabular Data workshop at EurIPS 2025.

source, missing data must be imputed to numerical values before employing statistical or machine learning models.



(a) Imputation Accuracy (↑) (b) Runtime (↓)

Figure 1: **Evaluation on real-world OpenML data: MissBench.** We compare TabImpute and TabImpute+ (ensembled method) with 11 other popular methods on MissBench. In panel (a), we plot the imputation accuracy (defined as 1 - normalized RMSE), which is calculated for each method, normalized within a dataset, and averaged across datasets and 13 missingness patterns. Error bars indicate 95% confidence intervals. In panel (b), we compare the runtime per table entry. Any method not labeled (GPU) is tested on a CPU because that method is not GPU-compatible. TabPFN on CPU is significantly slower, so we do not include it. See Sec. 2 for our exact computing specifications and Sec. 3 for accuracy score methodology.

Building on recent advances in tabular representation learning (Hollmann et al., 2023; Ye et al., 2025), we propose a pre-trained transformer specifically designed for the tabular missing-data problem that produces accurate and fast zero-shot imputations. TabPFN is a pre-trained transformer model for supervised learning that performs well across a variety of domains without any fine-tuning (Hollmann et al., 2025). The team behind TabPFN created an imputation method in their `tabpfn-extensions` Python package by using the TabPFN model in iterative column-wise imputation (available here[1]).

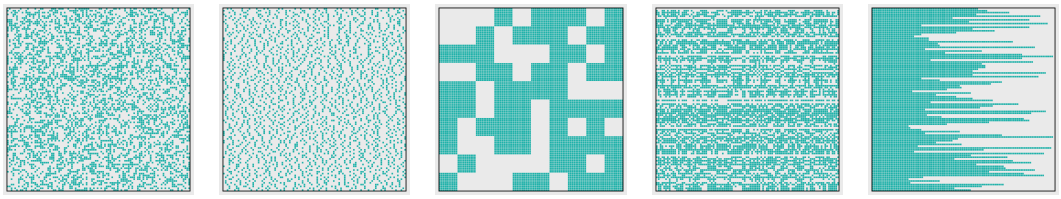


Figure 2: **Selection of synthetic missingness patterns implemented in MissBench.** Blue entries indicate observed values, and gray entries are unobserved.

The main contributions of this work can be summarized as follows:

- We introduce a novel, comprehensive test bench, denoted MissBench, using 42 real-world OpenML datasets and 13 missingness patterns (see Sec. 3 for details). Our benchmark builds on previous work, namely HyperImpute (Jarrett et al., 2022) comprising of 13 UC Irvine (UCI) datasets (Kelly et al., 2024), a subset of OpenML, with 3 missingness patterns, and GAIN (Yoon et al., 2018) comprising of 6 UCI datasets with 1 missingness pattern.
- Next we introduce a state-of-the-art method in following steps:
  - First, we propose a new entry-wise missing data featurization (see Sec. 2.1 for details). Notably, this new featurization, when directly used with the pre-trained TabPFN model of Hollmann et al. (2023), provides a significant improvement in speed and accuracy over TabPFN. We call this method EWF-TabPFN.

[1] `https://github.com/PriorLabs/tabpfn-extensions/blob/main/src/tabpfn_extensions/unsupervised/unsupervised.py`

– Next, we develop a synthetic data generation pipeline to create training datasets with missing values covering a wide range of missingness patterns (see App. A.2, App. A.7, and Tab. 6 for details). We pre-train a TabPFN architecture on this pipeline with our new featurization—we call this method TabImpute. This model provides an improvement over EWF-TabPFN across several missingness patterns.
– Finally, we introduce an adaptive ensembling imputation method, denoted TabImpute+, to combine the power of TabImpute and EWF-TabPFN (see App. A.3 for details). This allows us to leverage the underlying world-knowledge of EWF-TabPFN and domain-knowledge for missing data from TabImpute to yield accurate imputations across heterogeneous datasets.

Our code and implementation details for all our contributions above can be accessed on GitHub.[2]

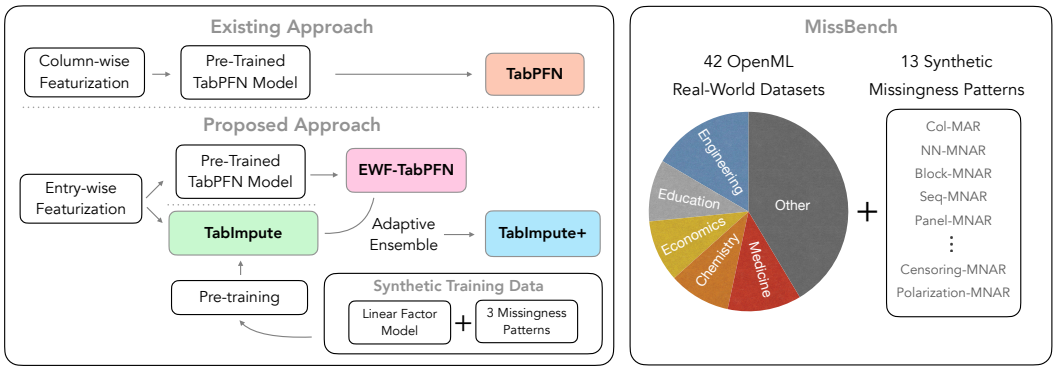


Figure 3: **Overview of our contributions.** The first row demonstrates TabPFN's imputation method, which performs iterative column-by-column imputation. We build on this by introducing an entry-wise featurization, as shown in the second row. We create a new synthetic data-generator for missingness data to train our model, TabImpute, shown in green (App. A.2 and App. A.5, respectively). Lastly, we ensemble TabImpute with TabPFN's model using our features to create TabImpute+ (App. A.3). We adaptively evaluate all the imputers on the comprehensive and rich set of OpenML datasets with many missingness patterns applied (Sec. 3).

2 Training TabImpute on Synthetic Data

We develop a featurization for tabular missing data imputation that enables better utilization of TabPFN's architecture, create a synthetic data generation pipeline across diverse missingness patterns, and employ an adaptive training algorithm to balance performance across all patterns. Training used 8 H200 GPUs and an Intel Xeon Platinum 8592+ CPU over approximately one week, processing 25 million synthetic tables. Our model matches TabPFN's size and runs on CPU-only systems. Evaluation used 1 H200 GPU. For details on our training and ensembling approaches, see App. A.5 in the Appendix.

2.1 Entry-wise Featurization and Architecture

We recast missing data imputation as supervised learning to leverage TabPFN's architecture and enable parallel GPU computation of missing entries. For each dataset (training point), let X^* be the complete matrix with m rows and n columns, Ω be the set of missing entry indices, and X be the matrix with induced missingness:

$$X_{ij} = \begin{cases} X^*_{ij} & \text{for} \quad (i,j) \in [m] \times [n] \setminus \Omega \\ \star & \text{otherwise.} \end{cases}$$

where $\star$ denotes a missing entry. Let $\Omega_{\text{obs}} = [m] \times [n] \setminus \Omega$. Our feature matrix construction adds $(i \oplus j \oplus X_{i,:} \oplus X_{:,j})$ for each entry $i, j \in [m] \times [n]$, where $X_{i,:}$ denotes the i-th row, $X_{:,j}$ the j-th

[2] `https://anonymous.4open.science/r/tabular-6F65/README.md`

column, and $\oplus$ concatenation. Each row's target is $y_{ij} = X^*_{ij}$. During pre-training, we train the model to predict target values for all $(i, j) \in \Omega$. This procedure creates a feature matrix of size $nm \times (n + m)$. This featurization captures all necessary information for each cell through its row and column context while enabling parallel computation of missing entries on GPUs. Although the input matrix size increases, parallelization gains outweigh this cost.

3 Results on OpenML Datasets: MissBench

To evaluate TabImpute against other methods, we introduce MissBench: a missing-data imputation benchmark using 42 OpenML (Vanschoren et al., 2013) tabular datasets with 13 synthetic missingness patterns. For every dataset and missingness pattern, we test each method's ability to impute masked values. The 42 OpenML datasets span domains such as medicine, engineering, and education. The missingness patterns include 1 `MCAR` pattern, 1 `MAR` pattern, and 11 `MNAR` patterns. We provide details for the `MNAR` patterns in App. A.7.

Imputation Accuracy. To ensure a fair comparison across datasets with different scales and inherent difficulties, we report a normalized accuracy score. In particular, for each dataset and missingness pattern, we first calculate the standard Root Mean Squared Error (RMSE) for every imputation method as $\left(\frac{1}{|\Omega|}\sum_{(i,j)\in\Omega}\left(X_{ij}^{\text{true}} - X_{ij}^{\text{imputed}}\right)^2\right)^{1/2}$, where Ω denotes the set of missing entries. We then perform a min-max normalization on these RMSE scores across all methods for that specific task:

$$\text{Normalized RMSE} = \frac{\text{RMSE}_{\text{method}} - \min(\text{RMSE}_{\text{all methods}})}{\max(\text{RMSE}_{\text{all methods}}) - \min(\text{RMSE}_{\text{all methods}})}$$

This normalization maps the best-performing method to 0 and the worst to 1. Finally, we define our *Imputation Accuracy* as $1 -$ Normalized RMSE, where higher values indicate better performance.

Tab. 1 presents results for each missingness pattern as well as overall performance. TabImpute+ achieves the best overall performance and for nearly all individual patterns. For completeness, we list the performance of methods not shown in the table in App. A.8 with non-normalized RMSE examples in Tab. 8. TabImpute+ performs best under high missingness conditions (Fig. 4, App. A.6), which is expected since it leverages generative pre-training rather than relying solely on available dataset information like discriminative methods.

Table 1: Imputation Accuracy ± Standard Deviation by Missingness Pattern.
We train on the patterns above the dashed line.

Pattern	TabImpute+	HyperImpute	OT	MissForest
`MCAR`	0.821 ± 0.157	0.804 ± 0.205	0.856 ± 0.114	**0.867 ± 0.148**
`NN-MNAR`	**0.880 ± 0.126**	0.762 ± 0.240	0.815 ± 0.139	0.819 ± 0.188
`Self-Masking-MNAR`	0.707 ± 0.279	**0.717 ± 0.257**	0.593 ± 0.246	0.666 ± 0.272
`Col-MAR`	**0.860 ± 0.178**	0.819 ± 0.263	0.756 ± 0.262	0.776 ± 0.251
`Block-MNAR`	**0.908 ± 0.168**	0.873 ± 0.178	0.857 ± 0.152	0.860 ± 0.153
`Seq-MNAR`	**0.905 ± 0.094**	0.862 ± 0.193	0.892 ± 0.094	0.829 ± 0.203
`Panel-MNAR`	**0.791 ± 0.329**	0.515 ± 0.368	0.540 ± 0.374	0.526 ± 0.371
`Polarization-MNAR`	**0.885 ± 0.137**	0.620 ± 0.263	0.804 ± 0.125	0.559 ± 0.231
`Soft-Polarization-MNAR`	**0.864 ± 0.176**	0.705 ± 0.223	0.755 ± 0.195	0.670 ± 0.269
`Latent-Factor-MNAR`	**0.887 ± 0.121**	0.778 ± 0.231	0.844 ± 0.167	0.836 ± 0.152
`Cluster-MNAR`	**0.873 ± 0.135**	0.841 ± 0.183	0.828 ± 0.137	0.833 ± 0.141
`Two-Phase-MNAR`	0.855 ± 0.186	0.871 ± 0.200	0.809 ± 0.183	**0.878 ± 0.123**
`Censoring-MNAR`	0.594 ± 0.302	**0.797 ± 0.262**	0.599 ± 0.239	0.682 ± 0.249
Overall	**0.833 ± 0.213**	0.766 ± 0.259	0.765 ± 0.227	0.754 ± 0.248

4 Conclusion & Future Work

In this paper, we introduce a comprehensive benchmark for tabular missing data, MissBench, and a pre-trained transformer for the tabular missing data problem, TabImpute. We build on recent work in tabular representation learning by adapting TabPFN's architecture and training pipeline for the missing data setting. While we train purely on synthetic data, we are able to impute entries accurately on real-world OpenML data for a comprehensive set of missingness patterns, showcasing our model's ability to generalize to unseen domains. We open-source our model architecture, weights, and our training and evaluation code (available here[3]).

Note that since TabImpute uses the same architecture as TabPFN, any improvements to TabPFN's architecture can be immediately ported to TabImpute. Finally, since we use a PFN architecture, we output a distribution for each missing entry and can sample from this distribution for multiple imputation (Rubin, 2018). In future work, we plan on (i) exploring further training on more complex missingness patterns and data-generating processes, (ii) enhancing our method to support categorical data, (iii) extending our evaluation set to causal inference settings, which can be modeled as missing-data problems (Agarwal et al., 2023), (iv) improving the architecture to scale to larger datasets, and (v) utilizing our method for multiple imputation.

[3] `https://anonymous.4open.science/r/tabular-6F65/README.md`

References

Anish Agarwal, Munther Dahleh, Devavrat Shah, and Dennis Shen. Causal matrix completion. In *The thirty sixth annual conference on learning theory*, pp. 3821–3826. PMLR, 2023.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

Jushan Bai and Serena Ng. Determining the number of factors in approximate factor models. *Econometrica*, 70(1):191–221, 2002. doi: https://doi.org/10.1111/1468-0262.00273. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/1468-0262.00273.

Emmanuel Candes and Benjamin Recht. Exact matrix completion via convex optimization. *Communications of the ACM*, 55(6):111–119, 2012.

Evelyn Fix and J. L. Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review / Revue Internationale de Statistique*, 57(3):238–247, 1989. ISSN 03067734, 17515823. URL http://www.jstor.org/stable/1403797.

Susobhan Ghosh, Raphael Kim, Prasidh Chhabria, Raaz Dwivedi, Predrag Klasnja, Peng Liao, Kelly Zhang, and Susan Murphy. Did we personalize? assessing personalization by an online reinforcement learning algorithm using resampling. *Machine learning*, 113(7):3961–3997, 2024.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

Trevor Hastie, Rahul Mazumder, Jason D Lee, and Reza Zadeh. Matrix completion and low-rank svd via fast alternating least squares. *The Journal of Machine Learning Research*, 16(1):3367–3402, 2015.

Graeme Hawthorne and Peter Elliott. Imputing cross-sectional missing data: comparison of common techniques. *Australian & New Zealand Journal of Psychiatry*, 39(7):583–590, 2005.

Jeff Heaton. Ian goodfellow, yoshua bengio, and aaron courville: Deep learning: The mit press, 2016, 800 pp, isbn: 0262035618. *Genetic programming and evolvable machines*, 19(1):305–307, 2018.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=cp5PvcI6w8_.

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, 2025.

Daniel Jarrett, Bogdan C Cebere, Tennison Liu, Alicia Curth, and Mihaela van der Schaar. Hyperimpute: Generalized iterative imputation with automatic model selection. In *International Conference on Machine Learning*, pp. 9916–9937. PMLR, 2022.

Alexia Jolicoeur-Martineau, Kilian Fatras, and Tal Kachman. Generating and imputing tabular data via diffusion and flow-based gradient-boosted trees. In *International conference on artificial intelligence and statistics*, pp. 1288–1296. PMLR, 2024.

Markelle Kelly, Rachel Longjohn, and Kolby Nottingham. The uci machine learning repository. https://archive.ics.uci.edu, 2024. Accessed: 2025-09-19.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. doi: 10.1109/MC.2009.263.

Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

Pierre-Alexandre Mattei and Jes Frellsen. Miwae: Deep generative modelling and imputation of incomplete data sets. In *International Conference on Machine Learning*, pp. 4413–4423. PMLR, 2019.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=KSugKcbNf9.

Boris Muzellec, Julie Josse, Claire Boyer, and Marco Cuturi. Missing data imputation using optimal transport. In *International Conference on Machine Learning*, pp. 7130–7140. PMLR, 2020.

Patrick Royston and Ian R White. Multiple imputation by chained equations (mice): implementation in stata. *Journal of statistical software*, 45:1–20, 2011.

Donald B Rubin. Multiple imputation. In *Flexible imputation of missing data, second edition*, pp. 29–62. Chapman and Hall/CRC, 2018.

Daniel J. Stekhoven and Peter Bühlmann. Missforest—non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1):112–118, 10 2011.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

Stef Van Buuren. *Flexible imputation of missing data*, volume 10. CRC press Boca Raton, FL, 2012.

Stef van Buuren and Karin Groothuis-Oudshoorn. Mice: Multivariate imputation by chained equations in r. *Journal of Statistical Software*, 45(3):1–67, 2011. doi: 10.18637/jss.v045.i03. URL https://www.jstatsoft.org/index.php/jss/article/view/v045i03.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198. URL http://doi.acm.org/10.1145/2641190.2641198.

Han-Jia Ye, Si-Yang Liu, and Wei-Lun Chao. A closer look at tabpfn v2: Strength, limitation, and extension. *arXiv preprint arXiv:2502.17361*, 2025.

Jinsung Yoon, James Jordon, and Mihaela van der Schaar. Gain: Missing data imputation using generative adversarial nets. In *International Conference on Machine Learning (ICML)*, 2018.

A Appendix

Here we present the rest of the missingness patterns we tested on, tables with further results, the methods we tested against, and the OpenML datasets we evaluated on.

A.1 Data Generation with Linear Factor Models

We generate data using a simple linear factor model (LFMs) (Bai & Ng, 2002). LFMs are commonly used in matrix completion literature to prove error bounds for matrix completion algorithms (Koren et al., 2009; Candes & Recht, 2012). This family of models generates a data matrix $Y \in \mathbb{R}^{m \times n}$ by assuming the data lies on or near a low-dimensional subspace. The simplest case generates the data matrix Y as the inner product of two lower-rank latent factor matrices, $U \in \mathbb{R}^{m \times k}$ and $V \in \mathbb{R}^{n \times k}$, where $k \ll n, m$ is the rank:

$$Y = UV^T.$$

To generate diverse datasets, the latent vectors (rows of U and V) are sampled from a variety of distributions, including Gaussian, Laplace, Student's t, spike-and-slab (a mixture of a Dirac delta at zero and a Gaussian), and Dirichlet.

When training, we experimented with several classes of data-generating processes, including matrices from nonlinear factor models and structural causal models (SCM) similar to the ones used in TabPFN. However, we found that the model trained on linear factor models performed the best. See Tab. 5 in App. A.8 for results demonstrating this.

Architecture. We use TabPFN's base architecture with one modification: removing the attention mask to allow training points to attend to test points. Since our train/test set sizes vary randomly with missingness patterns (unlike TabPFN's controlled synthetic generation), we remove the mask to enable parallel batch training. TabPFN's mask prevents train points from seeing test feature distributions, which is important in general supervised learning. However, our *test* set is created using data already available to the observed points, thus alleviating any data-leakage concerns.

A.2 Synthetic Training Data Generation

We generate tens of millions of datasets with missing values to train our model through a two-step process: first, generating underlying data, then introducing missingness patterns on top. See the App. A.1 in the Appendix for details. For details on our training pipeline, see App. A.5 in the Appendix.

A.3 Adaptive Ensembling

Our final method, TabImpute+, includes two layers of ensembling: (i) permutation-based preprocessing techniques and (ii) adaptive weighted average of TabImpute and EWF-TabPFN. Note that TabPFN's regression class includes several ensembling techniques as well. The first level of ensembling performs random permutations of the rows and columns before running these permuted matrices through TabImpute, reversing the permutations, and then averaging the predictions across the runs. We used 4 independent permutations for our final method.

Ensembling with EWF-TabPFN enables our model to accurately capture features common in linear factor models while also using the nonlinear function approximation power of EWF-TabPFN. Both TabImpute and EWF-TabPFN can predict outputs for the observed values as well, enabling us to calculate an optimal weighting between them that minimizes the mean square error for their observed-entry outputs. This is a one-dimensional quadratic minimization problem and a simple, closed-form solution immediately follows: let $\hat{x}^{(1)}, \hat{x}^{(2)}, \hat{x}^{obs} \in \mathbb{R}^k$ where k is the number of observed entries, $\hat{x}^{(1)}$ is the prediction from our model, $\hat{x}^{(2)}$ is the prediction using EWF-TabPFN, and $\hat{x}^{obs}$ are the observed values. Then, the optimal weight solves: $\min_w ||\hat{x}^{obs} - (w\hat{x}^{(1)} + (1-w)\hat{x}^{(2)})||_2^2$. This has a unique solution: $w^* = (\hat{x}^{obs} - \hat{x}^{(2)})^T(\hat{x}^{(1)} - \hat{x}^{(2)})/||\hat{x}^{(1)} - \hat{x}^{(2)}||^2$. Since this is calculated based on the observed values, this weighting adapts at inference time and is very fast to calculate.

A.4 Missingness Patterns for Training and Evaluation

After generating a complete data matrix $X^* \in \mathbb{R}^{m \times n}$, we introduce missingness by applying a masking matrix $M \in \{0,1\}^{m \times n}$, where the entry value $M_{ij} = 1$ if and only if $(i,j) \in \Omega_{\text{obs}}$. To ensure TabImpute is robust and generalizable to the variety of ways data can be missing in real-world scenarios, we pre-train on a comprehensive stock of synthetic datasets with several missingness patterns. For convenience, we define $p_{ij} = \mathbb{P}(M_{ij} = 1)$, the propensity of each entry in X^*.

We include 13 different missingness patterns: 1 `MCAR`, 1 `MAR` pattern, and 11 `MNAR` patterns. For examples of these, see Fig. 2. `MNAR` patterns often stump standard imputation methods and yet are extremely common in the real world.

`MCAR`: `MCAR` missingness means the probability of an entry being missing, defined as its propensity, is constant across entries and independent from any other randomness. The missingness indicators M_{ij} are drawn i.i.d. from a Bernoulli distribution $M_{ij} \sim \text{Bern}(p)$ across all $(i,j) \in [m] \times [n]$ for some constant $p \in (0,1)$. This is the simplest form of missingness, but is unrealistic (Van Buuren, 2012).

`MAR`: For `MAR` missingness, the probability of an entry being missing depends only on the observed values X. In other words, the randomness in `MAR` can be explained by conditioning on observed factors. Additionally, every entry has a positive probability of being observed (i.e., $p_{ij} > 0$). We simulate `MAR` through column-wise `MAR`, denoted `Col-MAR`: we choose several columns as predictor columns and use those values to mask entries in other columns. This is similar to the `MAR` approach taken in Jarrett et al. (2022).

`MNAR` Patterns: For the most complex missingness class, `MNAR`, the probability of an entry being missing can depend on unobserved factors. Note that `MNAR` patterns are significantly more difficult to handle systematically, often requiring specialized methods for a specific kind of `MNAR` pattern (Van Buuren, 2012). Due to the flexibility of our entry-wise featurization, TabImpute can produce imputations for these highly complex scenarios, even when columns are completely missing, such as in panel-data missingness patterns. HyperImpute was tested on two `MNAR` patterns in the Appendix of Jarrett et al. (2022), one where values are further masked after an `MAR` pattern and another where values outside a certain range are masked. We build on this work by testing on 11 `MNAR` patterns (see App. A.7 and Tab. 6 for details). We implement a range of `MNAR` patterns to simulate plausible real-world scenarios. For example, we utilize the expressiveness of neural networks to create random propensity functions `MNAR` patterns, use bandit algorithms to induce column-adaptive missing patterns, simulate panel data missingness where some features are removed later, censoring where sensor readings fall outside a detectable range, and survey data artifacts like respondent polarization and skip-logic.

A.5 Training on Multiple Missingness Patterns

We train our model to predict unobserved values under several missingness patterns simultaneously. For the final TabImpute model, we trained only on `MCAR`, `NN-MNAR`, and `Self-Masking-MNAR` because we found our model generalized to the other methods well without including them explicitly in training. We use the prior-data fitted negative log likelihood (NLL) loss proposed in Müller et al. (2022). Like other PFNs with continuous numerical output, we use the Riemann distribution output also proposed in Müller et al. (2022). Since we can generate an unlimited amount of synthetic data, we do not reuse any synthetic data and only do one gradient pass per batch of datasets. This allows our model to learn the underlying data-generating process and missingness mechanisms without risk of memorization. We use a learning rate of 0.0001, a batch size of 64, and train on around 25 million synthetic datasets.

We use an adaptive algorithm to determine what proportion of missingness patterns we include in each batch as we train: Every s gradient steps, we recalculate the proportion of each missingness pattern by: (i) create new batches of each missingness pattern, (ii) calculate the model's loss value for each missingness pattern's batch, and (iii) recalculate proportions by running softmax (Heaton, 2018, Pgs. 180-184) over the loss values. This procedure adaptively down-weights any missingness type our model performs well on and up-weights any missingness type our model performs poorly on. For our pre-trained TabImpute model, we set $s = 50$ steps.

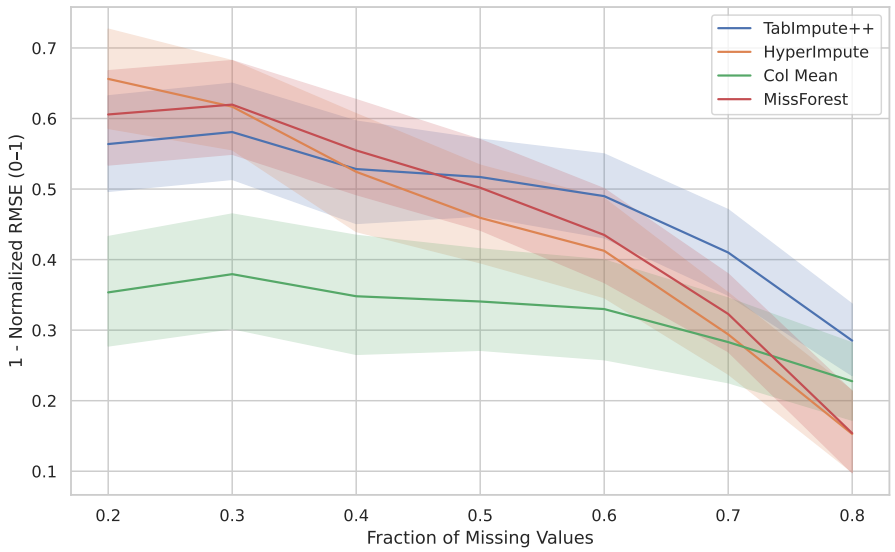


Figure 4: **Imputation accuracy versus fraction of missingness for `MCAR`.** TabImpute+ performs the best when missingness is higher because it is a generative model that fits to the data in context.

Remark: We had initially attempted to train the model sequentially on one missingness pattern at a time, but found that the network always experienced *catastrophic forgetting* (McCloskey & Cohen, 1989) irrespective of learning rate (i.e., it forgot how to handle the previous missingness patterns).

A.6 Additional tests

Next, we discuss when TabImpute+ does well and when HyperImpute and other methods do well. We found that an important factor in determining performance was the level of missingness. The probability of missingness can only be controlled precisely for `MCAR`. Thus, we show in Fig. 4 the performance of the top methods as we increase the missingness level.

Building on TabPFN. The TabPFN team's imputation extension predicts missing entries column-by-column using their pre-trained model. However, as shown in Fig. 1, this method has poor accuracy and slow runtime even on H200 GPUs. This motivated us to introduce a new entry-wise featurization that achieves improved performance using TabPFN's model, which we denote as EWF-TabPFN. We initially attempted fine-tuning TabPFN with our synthetic data but consistently encountered catastrophic forgetting, with performance degrading to match TabImpute, the model trained from scratch. Without access to TabPFN's original training data, proper data augmentation proved immensely challenging. Instead, we elected to ensemble TabImpute and EWF-TabPFN as TabImpute+, leveraging their complementary performance across benchmark patterns (Tab. 2). **Crucially, the performance scores in this table are re-normalized using only the results of these three methods.** This means the minimum and maximum RMSE values used for normalization are determined from this smaller set, leading to different numerical scores than in Tab. 1, which was normalized across all 12 competing methods. This approach offers a clearer view of the relative performance and complementarity within our proposed family of models, highlighting which components excel at different missingness patterns. (See App. A.8 for other test results.)

A.7 Details for `MNAR` Missingness Patterns

This section provides the mathematical and implementation details for each of the simulated `MNAR` missingness mechanisms that we implement and test. For each pattern, we define the mechanism by which the missingness mask M is generated, where $M_{ij} = 1$ if the value X_{ij} is observed and $M_{ij} = 0$ otherwise.

Table 2: Imputation Accuracy ± Standard Deviation for Zero-Shot Methods.

Pattern	TabImpute+	EWF-TabPFN	TabImpute
MCAR	0.349 ± 0.468	0.324 ± 0.465	**0.667 ± 0.477**
NN-MNAR	**0.551 ± 0.480**	0.458 ± 0.488	0.487 ± 0.488
Self-Masking-MNAR	0.945 ± 0.214	**0.952 ± 0.211**	0.048 ± 0.216
Col-MAR	**0.799 ± 0.391**	0.773 ± 0.414	0.220 ± 0.419
Block-MNAR	0.498 ± 0.474	0.437 ± 0.494	**0.559 ± 0.501**
Seq-MNAR	0.394 ± 0.482	0.363 ± 0.447	**0.619 ± 0.492**
Panel-MNAR	0.565 ± 0.476	**0.568 ± 0.476**	0.415 ± 0.499
Polarization-MNAR	0.775 ± 0.407	**0.788 ± 0.411**	0.214 ± 0.415
Soft-Polarization-MNAR	0.698 ± 0.422	**0.791 ± 0.383**	0.258 ± 0.439
Latent-Factor-MNAR	0.505 ± 0.490	0.467 ± 0.480	**0.524 ± 0.505**
Cluster-MNAR	0.428 ± 0.474	0.420 ± 0.487	**0.571 ± 0.501**
Two-Phase-MNAR	**0.591 ± 0.475**	0.560 ± 0.494	0.429 ± 0.501
Censoring-MNAR	0.891 ± 0.304	**0.902 ± 0.282**	0.098 ± 0.300
Overall	**0.614 ± 0.468**	0.600 ± 0.476	0.393 ± 0.487

A.7.1 Details for NN-MNAR

Description The pattern simulates a scenario where the propensity p_{ij} depends on the underlying matrix values X^* in an arbitrary manner. We achieve a comprehensive coverage of MNAR patterns by leveraging the expressiveness of neural networks.

Methodology One general form of MNAR can be described as follows: for all i and j, there exists some function f_{ij} on the true (hence unobserved) matrix X^* such that the propensity depends on X^* as follows: $p_{ij}(X^*) = \mathbb{P}(M_{ij} = 1|X^*) = f_{ij}(X^*)$. By leveraging the expressiveness of neural networks, we propose a neural-net-based MAR pattern generator (NN-MNAR) that is designed to approximate arbitrary propensities characterized by functions f_{ij}.

Implementation Details For fixed indices i and j, NN-MNARconstructs the propensity p_{ij} in a two-step procedure. First, we randomly collect a subset of values from the matrix X^* and flatten them as a vector, say $X^*(i, j)$; we do this by first randomly generating a neighborhood $\mathbf{N}_{ij} \subset [m] \times [n]$, then the entries in the neighborhood $X^*_{st}, (s, t) \in \mathbf{N}_{ij}$ constitute the entries of the vector $X^*(i, j)$. Second, a neural-net function $g_{ij} : \mathbb{R}^{|\mathbf{N}_{ij}|} \to [0, 1]$ is constructed by randomly initializing the number of layers, depth, weight, and bias. At each training step, the random neighborhood $\mathbf{N}_{ij}$ and the random neural-net g_{ij} collectively defines the propensity $p_{ij} = g_{ij}(X^*(i, j))$ from which MNAR missingness patterns are generated $M_{ij} \sim \text{Bern}(p_{ij})$.

A.7.2 Details on Seq-MNAR

Description This pattern simulates a scenario where masking matrix values $M_{ij} \in \{0, 1\}$ for each column j are adaptively chosen depending on the information up to column $j - 1$ (i.e., regard columns as time). Specifically, we employ variants of bandit algorithms Lattimore & Szepesvári (2020) while regarding the binary masking matrix values as the two arms. Such patterns commonly arise in sequential experiments Ghosh et al. (2024).

Methodology The true matrix X^* is transformed to constitute the reward. For each designated column j, one of the following bandit algorithm utilizes the all the information of X^* and M up to column $j - 1$ and chooses one of the two arms $\{0, 1\}$ via one of the following algorithms: ε-greedy, upper-confidence bound (UCB), Thompson-sampling (Thompson, 1933) or gradient bandit.

Implementation Details We generate exogenous Gaussian noise and add it to the true matrix X^* and regard X^* as the reward for arm 0 and its noisy version as the reward for arm 1. Then, starting from the first column with multiple rows as multiple agents, we randomly initiate (with random configurations) one of the four algorithms Lattimore & Szepesvári (2020): ε-greedy, Upper Confidence Bound (UCB) (Auer et al., 2002), Thompson sampling with random configurations

(Thompson, 1933). Further, we have the option to randomly mix pooling techniques Ghosh et al. (2024) on top of any of the four algorithms.

A.7.3 `Self-Masking-MNAR`

Description: This pattern simulates a scenario where the probability of a value being missing is a direct function of the value itself. This pattern can be regarded as a special case of `NN-MNAR`. A common real-world example includes individuals with very high incomes being less likely to report their salary on a survey.

Methodology: For each designated target column j, the probability of an entry (i, j) being missing is determined by a logistic function of its value. The relationship is defined as $\mathbb{P}(M_{ij} = 0|X^*_{ij}) = \sigma(\alpha \cdot X^*_{ij} + \beta_0)$ where $\sigma(z) = (1 + e^{-z})^{-1}$ is the sigmoid function.

Implementation Details: A random coefficient α is chosen from the set $\{-2, -1, 1, 2\}$ to introduce variability in the direction and magnitude of the value's effect on its missingness probability. The bias term β_0 is calibrated to achieve a target missingness proportion, p.

A.7.4 `Censoring-MNAR`

Description: This pattern models missingness that arises from the practical/physical limits of measurement equipment, where values below a lower detection limit (left-censoring) or above an upper detection limit (right-censoring) are not recorded. We would expect to see such missingness in sensor or biological assay datasets where test equipment can only detect biomarker levels above/below certain thresholds.

Methodology: For each column j, a censoring direction (left or right) is chosen with equal probability. A cutoff value is determined based on a specified quantile, q_{censor}, of the set of currently observed (non-missing) values in that column. Let $X^*_{:,j}$ denote the set of observed values in column j, i.e., $X^*_{:,j} = \{X_{ij} \mid M_{ij} = 1\}$.

- **Left-Censoring:** All values in column j that are less than the q_{censor}-th quantile of the observed values in that same column are set to missing. The threshold is a single scalar value calculated from the column's observed data.
$$M_{ij} = 0 \quad \text{if} \quad X_{ij} < \text{quantile}(X^*_{:,j}, q_{\text{censor}})$$
- **Right-Censoring:** All values in column j that are greater than the $(1 - q_{\text{censor}})$-th quantile of the observed values in that column are set to missing. The threshold is a single scalar value.
$$M_{ij} = 0 \quad \text{if} \quad X_{ij} > \text{quantile}(X^*_{:,j}, 1 - q_{\text{censor}})$$

Implementation Details: The choice between left- and right-censoring is made randomly for each column with a probability of 0.5 for each. We introduce a hyperparameter q_{censor} for the censoring quantile that controls the fraction of data to be censored from either tail of the distribution. For our evaluation, we use $q_{\text{censor}} = 0.25$.

A.7.5 `Panel-MNAR`

Description: This pattern simulates participant dropout in longitudinal or panel data studies, where once a subject (row) drops out at a specific time point, all their subsequent data is missing. This is a common occurrence in clinical trials or long-term studies/surveys.

Methodology: The columns of the data matrix X^* are assumed to represent ordered time points $t = 0, 1, ..., T - 1$. For each subject (row) i, a random dropout time $t_{0,i}$ is sampled. All observations for that subject from time $t_{0,i}$ onwards are masked as missing.
$$M_{ij} = 0 \quad \forall j \geq t_{0,i}$$

Implementation Details: For each row i, the dropout time $t_{0,i}$ is sampled uniformly from the range of possible time steps, i.e., $t_{0,i} \sim \text{Unif}\{1, ..., T - 1\}$.

A.7.6 `Polarization-MNAR`

Description: Values falling in the middle of a feature's distribution are preferentially removed, simulating survey non-response from individuals with moderate opinions. This is implemented by setting values between the q-th and $(1-q)$-th quantiles to missing. A "soft" version makes the observation probability proportional to the value's distance from the median. Such patterns are most commonly seen in political polls, where moderate voters are less likely to respond while those with extreme views are more likely to respond, or in product reviews, where only very satisfied or very dissatisfied customers leave ratings.

Hard Polarization Methodology: For each column j, values falling between two quantiles are deterministically masked. The quantiles are calculated using only the set of currently observed (non-'NaN') values in that column. Let $X^*_{:,j}$ denote this set of observed values, i.e., $X^*_{:,j} = \{X_{ij} \mid M_{ij} = 1\}$. The lower and upper thresholds, L_j and H_j, are defined as:

$$L_j = \text{quantile}(X^*_{:,j}, q_{\text{thresh}})$$

$$H_j = \text{quantile}(X^*_{:,j}, 1 - q_{\text{thresh}})$$

An entry X_{ij} is then masked if its value falls between these two scalar thresholds:

$$M_{ij} = 0 \quad \text{if} \quad L_j < X_{ij} < H_j$$

Soft Polarization Methodology: The probability of a value being observed is made proportional to its normalized absolute distance from the column's median, μ_j. This creates a softer, probabilistic version of the polarization effect. The missing probability is given by:

$$\mathbb{P}(M_{ij} = 0) = \epsilon + (1 - 2\epsilon)\frac{|X^*_{ij} - \mu_j|^\alpha}{\max_k(|X^*_{kj} - \mu_j|^\alpha)}$$

Implementation Details: For the hard polarization pattern, we introduce a hyperparameter $\boldsymbol{q}_{\text{thresh}}$ for the threshold quantile that defines the central portion of the distribution to be masked. For the soft polarization pattern, we have an exponent parameter α that controls the intensity of the polarization effect. Higher values of α make the observation probability more sensitive to deviations from the median. In the soft version, we also have a baseline probability ϵ that ensures even values at the median have a non-zero chance of being observed.

A.7.7 `Latent-Factor-MNAR`

Description: This pattern generates a complex missingness structure where the probability of an entry being missing depends on unobserved (latent) characteristics of both its row and its column. This is common in recommender systems, where a user's decision to rate an item depends on latent user preferences and item attributes. Other real-world examples include job application screenings where certain combinations of demographic factors influence whether candidates complete applications, independent of their qualifications, or online dating profiles where cultural background affects disclosure of personal information.

Methodology: The probability of an entry (i, j) being observed is modeled using a low-rank bilinear model. The observation probability is given by the sigmoid of a dot product of latent factors plus bias terms:

$$\mathbb{P}(M_{ij} = 1) = \sigma(u_i^T v_j + b_i + c_j)$$

where $u_i \in \mathbb{R}^k$ and $v_j \in \mathbb{R}^k$ are k-dimensional latent vectors for row i and column j, and b_i and c_j are scalar biases for the row and column, respectively.

Implementation Details: We specify the rank k that defines the dimensionality of the latent space, sampled as an integer. The elements of the latent factor matrices $U \in \mathbb{R}^{N \times k}$ and $V \in \mathbb{R}^{D \times k}$, and the bias vectors $b \in \mathbb{R}^N$ and $c \in \mathbb{R}^D$, are sampled independently from a standard normal distributions.

A.7.8 `Cluster-MNAR`

Description: This pattern induces missingness based on latent group-level characteristics. Rows and columns are first assigned to discrete clusters, and each cluster has a random effect that uniformly influences the observation probability of all its members. This is useful for modeling data from grouped experiments, such as a clinical trial where patients (rows) from a specific hospital (a given row cluster) and certain lab tests (a column cluster) might have systematically different rates of missingness due to local care protocols or equipment availability.

Methodology: The probability of an entry (i, j) being observed is determined by an additive model of random effects corresponding to the cluster assignments of its row i and column j. Denoting the row assignments by $C_R(i)$ and column assignments by $C_C(j)$, the observation probability is modeled as:

$$\mathbb{P}(M_{ij} = 1) = \sigma(g_{C_R(i)} + h_{C_C(j)} + \epsilon_{ij})$$

where:

- $\sigma(z) = (1 + e^{-z})^{-1}$ is the sigmoid function.
- $g_k \sim \mathcal{N}(0, \tau_r^2)$ is the random effect for row cluster k.
- $h_l \sim \mathcal{N}(0, \tau_c^2)$ is the random effect for column cluster l.
- $\epsilon_{ij} \sim \mathcal{N}(0, \epsilon_{\text{std}}^2)$ is an entry-specific noise term.

Implementation Details: For a matrix with N and D columns, row assignments $C_R(i)$ are draw uniformly from $\{0, ..., K_R - 1\}$ and column assignments $C_C(j)$ are drawn uniformly from $\{0, ..., K_C - 1\}$, where K_R and K_C are the total number of row and column clusters, respectively. The number of row clusters K_R, the number of column clusters K_C, and the standard deviation of the random effects (τ_r, τ_c, ϵ_{std}) are hyperparameters specified for the data generation process.

A.7.9 `Two-Phase-MNAR`

Description: This mechanism mimics multi-stage data collection where a subset of participants from an initial survey (with “cheap" features) are selected for a more detailed follow-up survey (with “expensive" features). The selection for the second phase is dependent on the data collected in the first, making this an MNAR pattern. A specific example includes market research where basic demographics are collected from all participants, but detailed purchasing behavior is only gathered from a subset, with missingness related to income level.

Methodology: Let $\mathcal{F} = \{0, 1, ..., D - 1\}$ be the set of all column indices in the data matrix X. This set is randomly partitioned into a "cheap" subset $\mathcal{C} \subset \mathcal{F}$ and an "expensive" subset $\mathcal{E} \subset \mathcal{F}$, such that $\mathcal{C} \cup \mathcal{E} = \mathcal{F}$ and $\mathcal{C} \cap \mathcal{E} = \emptyset$. By design, features in the cheap set $\mathcal{C}$ are always observed.

The decision to collect the expensive features for a given row i is based on a logistic model applied to its cheap features. Let $X_{i,\mathcal{C}}$ denote the vector of values $\{X_{ij} \mid j \in \mathcal{C}\}$ for row i. A score is calculated for each row:

$$s_i = \text{normalize}(X_{i,\mathcal{C}}^T w)$$

where w is a vector of random weights and the ‘normalize‘ function applies z-score normalization to the resulting scores across all rows.

The probability that all expensive features are observed for row i is then given by:

$$\mathbb{P}(M_{ij} = 1 \text{ for all } j \in \mathcal{E}) = \sigma(\alpha + \beta \cdot s_i)$$

If the expensive features for row i are not observed (based on the probability above), then all of its values in the expensive columns are masked as missing, i.e., $M_{ij} = 0$ for all $j \in \mathcal{E}$.

Implementation Details: A fraction of columns, e.g., 50%, are randomly assigned to be "cheap". The weight vector for the scoring model is sampled from a standard normal distribution, $w \sim \mathcal{N}(0, 1)$. Parameters α, β control the base rate and score-dependency of the observation probability. In our implementation, they are set to default values of $\alpha = 0$ and $\beta = 2.0$.

A.8 Additional tables

Table 3: Other imputation methods

Name	Description
Column-wise mean (Hawthorne & Elliott, 2005)	Mean of columns
SoftImpute (Hastie et al., 2015)	Iterative soft thresholding singular value decomposition based on a low-rank assumption on the data
k-Nearest Neighbors (Fix & Hodges, 1989)	Row-wise nearest neighbors mean
HyperImpute (Jarrett et al., 2022)	Iterative imputation method optimizing over a suite of imputation methods
Optimal transport method (Muzellec et al., 2020)	Uses optimal transport distances as a loss to impute missing values based on the principle that two randomly drawn batches from the same dataset should share similar data distributions
MissForest (Stekhoven & Bühlmann, 2011)	Repeatedly trains a random forest model for each variable on the observed values to predict and fill in missing entries until convergence
ICE (van Buuren & Groothuis-Oudshoorn, 2011)	Imputation with iterative and chained equations of linear/logistic models for conditional expectations
MICE (Royston & White, 2011)	Handles missing data by iteratively imputing each incomplete variable using regression models that condition on all other variables
GAIN (Yoon et al., 2018)	Adapts generative adversarial networks (Goodfellow et al., 2020) where the generator imputes missing values and the discriminator identifies which components are observed versus imputed
MIWAE (Mattei & Frellsen, 2019)	Learns a deep latent variable model and then performs importance sampling for imputation
ForestDiffusion (Jolicoeur-Martineau et al., 2024)	Trains a diffusion model using XGBoost directly on incomplete tabular data and then fills in missing values with an adapted inpainting algorithm

Table 4: Imputation Accuracy ± Standard Deviation by Missingness Pattern (Remaining Methods)

Pattern	K-Nearest Neighbors	ICE	Col Mean	TabPFN
MCAR	0.762 ± 0.158	0.658 ± 0.273	0.473 ± 0.263	0.410 ± 0.257
Col-MAR	0.846 ± 0.183	0.832 ± 0.248	0.627 ± 0.299	0.553 ± 0.330
NN-MNAR	0.757 ± 0.168	0.643 ± 0.254	0.507 ± 0.266	0.437 ± 0.266
Block-MNAR	0.857 ± 0.184	0.813 ± 0.239	0.774 ± 0.241	0.572 ± 0.335
Seq-MNAR	0.889 ± 0.114	0.819 ± 0.240	0.778 ± 0.214	0.641 ± 0.290
Self-Masking-MNAR	0.623 ± 0.268	0.681 ± 0.328	0.292 ± 0.258	0.268 ± 0.248
Panel-MNAR	0.614 ± 0.400	0.584 ± 0.407	0.502 ± 0.389	0.442 ± 0.384
Polarization-MNAR	0.631 ± 0.199	0.547 ± 0.279	0.972 ± 0.059	0.914 ± 0.187
Soft-Polarization-MNAR	0.575 ± 0.266	0.723 ± 0.217	0.754 ± 0.207	0.598 ± 0.327
Latent-Factor-MNAR	0.776 ± 0.186	0.716 ± 0.268	0.601 ± 0.297	0.508 ± 0.316
Cluster-MNAR	0.784 ± 0.150	0.758 ± 0.279	0.559 ± 0.278	0.500 ± 0.297
Two-Phase-MNAR	0.904 ± 0.124	0.881 ± 0.208	0.666 ± 0.273	0.466 ± 0.329
Overall	0.751 ± 0.238	0.721 ± 0.289	0.625 ± 0.313	0.526 ± 0.334

Pattern	MIWAE	MICE	GAIN
MCAR	0.236 ± 0.274	0.316 ± 0.283	0.543 ± 0.256
Col-MAR	0.491 ± 0.315	0.521 ± 0.354	0.276 ± 0.354
NN-MNAR	0.234 ± 0.280	0.310 ± 0.325	0.400 ± 0.324
Block-MNAR	0.594 ± 0.267	0.491 ± 0.311	0.184 ± 0.317
Seq-MNAR	0.586 ± 0.289	0.478 ± 0.329	0.204 ± 0.311
Self-Masking-MNAR	0.209 ± 0.269	0.631 ± 0.348	0.456 ± 0.380
Panel-MNAR	0.374 ± 0.363	0.251 ± 0.325	0.279 ± 0.360
Polarization-MNAR	0.535 ± 0.193	0.143 ± 0.230	0.264 ± 0.283
Soft-Polarization-MNAR	0.546 ± 0.239	0.187 ± 0.245	0.432 ± 0.409
Latent-Factor-MNAR	0.358 ± 0.296	0.331 ± 0.312	0.301 ± 0.305
Cluster-MNAR	0.321 ± 0.277	0.317 ± 0.287	0.356 ± 0.311
Two-Phase-MNAR	0.521 ± 0.321	0.557 ± 0.313	0.218 ± 0.320
Overall	0.417 ± 0.314	0.378 ± 0.338	0.326 ± 0.343

Table 5: Imputation Accuracy ± Standard Deviation by Method (Zero-shot models)
Note that the normalized numbers are slightly different from Tab. 1 because we did not include every other zero-shot method in the normalization in Tab. 1.

Method	MCAR	NN-MNAR	Self-Masking-MNAR	Col-MAR
TabImpute+	0.749 ± 0.246	**0.847 ± 0.200**	0.929 ± 0.178	**0.883 ± 0.255**
EWF-TabPFN	0.746 ± 0.247	0.841 ± 0.201	**0.935 ± 0.179**	0.882 ± 0.254
TabImpute	**0.860 ± 0.222**	0.828 ± 0.230	0.246 ± 0.254	0.551 ± 0.350
TabImpute (MCAR then MAR)	0.751 ± 0.284	0.666 ± 0.355	0.208 ± 0.243	0.499 ± 0.354
TabImpute (More Heads)	0.690 ± 0.261	0.700 ± 0.233	0.227 ± 0.255	0.523 ± 0.321
EWF-TabPFN Fine-Tuned	0.306 ± 0.189	0.279 ± 0.220	0.406 ± 0.313	0.339 ± 0.338
TabPFN	0.066 ± 0.193	0.067 ± 0.194	0.162 ± 0.224	0.160 ± 0.254

Method	Block-MNAR	Seq-MNAR	Overall
TabImpute+	**0.804 ± 0.272**	0.672 ± 0.361	**0.814 ± 0.270**
EWF-TabPFN	0.795 ± 0.279	0.670 ± 0.358	0.811 ± 0.271
TabImpute	0.778 ± 0.288	**0.834 ± 0.214**	0.683 ± 0.342
TabImpute (MCAR then MAR)	0.722 ± 0.287	0.805 ± 0.277	0.608 ± 0.362
TabImpute (More Heads)	0.668 ± 0.258	0.729 ± 0.260	0.589 ± 0.317
EWF-TabPFN Fine-Tuned	0.385 ± 0.305	0.347 ± 0.326	0.344 ± 0.288
TabPFN	0.048 ± 0.138	0.126 ± 0.262	0.105 ± 0.218

Table 6: Synthetic Data Generation Parameters

Missingness Pattern	**Parameter Name**	**Symbol**	**Value**
MCAR	Missing probability	p	0.4
Col-MAR	Missing probability	p	0.4
NN-MNAR	Neighborhood size	$\lvert N_{ij} \rvert$	Variable
	Network layers	L	Random
	Network depth	d	Random
	Weight initialization	W	Random
	Bias initialization	b	Random
Self-Masking-MNAR	Coefficient set	α	$\{-2, -1, 1, 2\}$
	Target missing proportion	$p_{missing}$	Variable
Block-MNAR	Missing probability	p	0.4
	Matrix size N	N	100
	Matrix size T	T	50
	Row blocks	B_r	10
	Column blocks	B_c	10
	Convolution type	-	mean
Seq-MNAR	Missing probability	p	0.4
	Algorithm	-	epsilon_greedy
	Pooling	-	False
	Epsilon	ϵ	0.4
	Epsilon decay	γ	0.99
	Random seed	s	42
Panel-MNAR	No explicit hyperparameters (dropout time sampled uniformly)		
Polarization-MNAR	Threshold quantile	q_{thresh}	0.25
Soft-Polarization-MNAR	Polarization alpha	α	2.5
	Polarization epsilon	ϵ	0.05
Latent-Factor-MNAR	Latent rank (low)	k_{low}	1
	Latent rank (high)	k_{high}	5
Cluster-MNAR	Number of row clusters	K_R	5
	Number of column clusters	K_C	4
Two-Phase-MNAR	Cheap feature fraction	f_{cheap}	0.4
Censoring-MNAR	Censoring quantile	q_{censor}	0.25

Table 7: OpenML datasets

Dataset	Size	Domain	Description
EgyptianSkulls	150×5	Anthropology	Cranial measurements over time in Egypt
humans_numeric	75×15	Biology	Human body measurements
FacultySalaries	50×5	Education/Economics	University faculty salary data
SMSA	59×16	Demographics/Economics	U.S. metropolitan statistical area data
Student-Scores	56×13	Education	Student exam scores
analcatdata_election2000	67×15	Political science	2000 U.S. presidential election results
analcatdata_gviolence	74×9	Criminology	Gun violence statistics
analcatdata_olympic2000	66×12	Sports/Economics	Olympic results and country stats
baskball	96×5	Sports analytics	Basketball performance data
visualizing_hamster	73×6	Education/Toy	Example dataset for teaching
witmer_census_1980	50×5	Demographics	U.S. census microdata (1980)
MercuryinBass	53×10	Environmental chemistry	Mercury concentrations in fish
SolarPower	204×5	Energy/Engineering	Solar power output records
WineDataset	178×14	Chemistry/Oenology	Wine physicochemical properties
alcohol-qcm-sensor	125×15	Analytical chemistry	Alcohol detection sensor readings
benzo32	195×33	Chemistry/Toxicology	Benzodiazepine compound data
machine_cpu	209×7	Computer systems	Predicting CPU performance
pwLinear	200×11	Mathematics/Engineering	Piecewise linear regression benchmark
pyrim	74×28	Chemistry/Pharmacology	Pyrimethamine bioassay compounds
slump	103×10	Civil engineering	Concrete slump test properties
ICU	200×20	Medicine	Intensive care patient data
appendicitis_test	106×8	Medicine	Appendicitis diagnosis
appendicitis_test_edsa	106×8	Medicine	Educational appendicitis dataset
breast-cancer-coimbra	116×10	Medicine	Breast cancer diagnosis data
Rainfall-in-Kerala-1901-2017	117×18	Climate science	Rainfall time series in Kerala
pollution	60×16	Environmental science	Air pollution measurements
treepipit	86×10	Ecology	Bird habitat distribution
autoPrice	159×16	Business/Economics	Automobile pricing dataset
dataset_analcatdata_creditscore	100×7	Finance	Credit scoring dataset
Swiss-banknote-conterfeit-detection	200×7	Finance/Fraud	Banknote authenticity classification
Glass-Classification	214×10	Forensics/Materials	Glass chemical composition (forensics)
chatfield_4	235×13	Statistics/Time series	Textbook time series data (Chatfield)
chscase_vine1	52×10	Agriculture/Statistics	Vine growth study
edm	154×18	Education	Student learning performance
metafeatures	75×32	Meta-learning	Dataset-level features
rabe_131	50×6	Chemistry/Benchmark	Spectroscopy regression dataset
rabe_148	66×6	Chemistry/Benchmark	Spectroscopy regression dataset
rabe_265	51×7	Chemistry/Benchmark	Spectroscopy regression dataset
sleuth_case1201	50×7	Statistics/Education	Applied regression textbook data
sleuth_ex1605	62×6	Statistics/Education	Applied regression textbook data
wisconsin	194×33	Medicine	Wisconsin breast cancer dataset

Table 8: Non-normalized RMSE values for MCAR pattern by dataset. Dataset columns are standardized based on observed values (mean 0, variance 1).

Dataset	TabImpute+	HyperImpute	MissForest	OT
EgyptianSkulls	**0.917**	0.961	0.946	0.966
FacultySalaries	0.693	0.811	**0.567**	0.572
Glass-Classification	0.937	**0.810**	0.886	0.902
ICU	**1.045**	1.137	1.112	1.045
MercuryinBass	1.373	1.377	**1.343**	1.359
Rainfall-in-Kerala-1901-2017	0.942	**0.902**	0.950	0.958
SMSA	0.813	0.939	**0.786**	0.807
SolarPower	**0.801**	0.873	0.888	0.914
Student-Scores	**0.391**	0.441	0.435	0.477
Swiss-banknote-conterfeit-detection	0.976	0.856	**0.730**	0.786
WineDataset	0.928	0.803	0.797	**0.789**
alcohol-qcm-sensor	0.590	0.580	**0.517**	0.560
analcatdata_election2000	**0.544**	0.662	0.615	0.722
analcatdata_gviolence	0.795	0.801	0.877	**0.794**
analcatdata_olympic2000	**1.867**	1.922	1.895	1.945
appendicitis_test	0.860	**0.650**	0.792	0.753
appendicitis_test_edsa	0.668	**0.532**	0.569	0.586
autoPrice	0.834	0.762	**0.702**	0.734
baskball	1.027	1.100	1.109	**1.017**
benzo32	1.055	1.055	1.053	**0.950**
breast-cancer-coimbra	**1.058**	1.183	1.098	1.104
chatfield_4	**0.475**	0.531	0.492	0.522
chscase_vine1	**0.808**	0.996	0.833	0.945
dataset_analcatdata_creditscore	**1.063**	1.131	1.122	1.065
divorce_prediction	0.525	0.528	**0.467**	0.493
edm	0.763	**0.553**	0.566	0.570
humans_numeric	**0.948**	0.970	0.991	1.011
machine_cpu	1.037	0.956	**0.858**	0.962
metafeatures	2.263	**2.090**	2.225	2.263
pollution	1.967	2.065	**1.837**	1.945
pwLinear	**1.075**	1.356	1.356	1.092
pyrim	0.877	0.863	0.859	**0.842**
rabe_131	**0.790**	1.003	1.031	0.997
rabe_148	0.984	1.017	**0.958**	1.045
rabe_265	1.131	1.298	**1.090**	1.106
sleuth_case1201	1.054	**0.939**	0.963	0.998
sleuth_ex1605	1.087	1.257	1.071	**0.969**
slump	0.871	**0.650**	0.805	0.787
treepipit	1.264	1.169	1.172	**1.167**
visualizing_hamster	0.872	0.905	**0.750**	0.774
wisconsin	0.777	**0.685**	0.783	0.817
witmer_census_1980	0.704	**0.616**	0.637	0.653