Appendix 1 Technical Background

In this section, we provide technical background on concepts leveraged within METRIK.

1.1 Imputation Algorithms

Simple methods for imputing clinical time series include relying on simple statistics (e.g., mean) (9) or last observation carried forward (1) but these are unable to model complex relationships over variables in multivariate datasets (9). Methods for learning estimators that leverage the cross-sectional and longitudinal correlations have been developed, demonstrating improved performance over these simple methods. Examples of such methods including 3D-MICE (51) and machine learning methods like gradient boosting that use features including global patient summary statistics and contextual information (e.g., measurements from other features and previous timepoints) to predict the missing measurement (22). These estimators rely on hand-crafted features, however, and prior studies have demonstrated that performance is sensitive to the choice of the feature set (52).

Given the challenges with designing a good feature set, an alternative is to use artificial neural networks for automated feature learning. Recent work has demonstrated the viability of learning such models for imputation of temporal datasets (18,21,53). Specifically, studies have demonstrated that the Transformer architecture (54,55), designed to model long-range dependencies using attention mechanisms, empirically ranks the top amongst various machine learning algorithms across various missingness mechanisms and rates on various datasets (53), including clinical ones (18). The MTSIT framework (18), achieves this high performance by implementing a model with a Transformer-based encoder and a linear decoder and training the model with the masked autoencoding objective, in which several elements from the input sequence are masked out (i.e., set to 0) and the decoder is tasked to predict the values of these elements (55). The specific training loss is shown in Eq. 1, where denotes the predicted values, X denotes the ground truth, and M denotes the set of indices within the tensor that have been masked out.

$$\mathcal{L}(\hat{\mathbf{X}}, \mathbf{X}) = \frac{1}{|M|} \sum_{(i,j,k) \in M} \left(\hat{\mathbf{X}}_{ijk} - \mathbf{X}_{ijk} \right)^2 \quad [5]$$

1.2 Search Algorithms and the Differentiable Mask Layer

Various search algorithms have been devised to find solutions in high-dimensional spaces (i.e., with more than 100 dimensions), including Bayesian Optimization (BO) (56), population-based methods (e.g., genetic algorithms) (56), and gradient-based methods (19). While standard BO does not scale well, high-dimensional variants can improve its scalability but impose assumptions on the underlying data structure that would require domain knowledge to justify (57). In contrast, population-based methods like genetic algorithms do not impose assumptions on the data structure and perform better than BO in practice (56) but are computationally intense owing to the iterative evaluation and adaptation of a large pool of candidate solutions (58).

In practice, gradient-based search finds solutions faster (19); however, is not applicable to solutions defined over discrete search spaces, such as a binary mask. One work (20) devised a method for differentiating over a binary mask. Specifically, the mask is represented by the Gumbel-sigmoid distribution, which instantiates the reparametrization trick to enable differentiation over the parameters of the distribution (59). The soft mask is then binarized using the threshold operator and passed through a straight-through estimator to enable differentiation through the thresholding step. The study (20) used the mask to identify task-specific subset of weights within a pre-trained neural network by parametrizing the mask over the entire weight space and applying the mask to the pre-trained, frozen neural network. The mask is learned by optimizing it on the task of interest that the neural network was trained on and penalizing its sparsity. The study validated the mask learning algorithm by demonstrating that the subnetwork parametrized by the learned mask maintains performance on test splits with similar distribution as the train splits but suffers drops in performance when evaluated on out-of-distribution splits.

Appendix 2 Implementation Details

In this section, we review implementation details related to METRIK and describe the computational resources used in our experiments.

METRIK

We set the size of the pilot study $n_{s,pilot}$ to 60, the recommended pilot study size for standard RCTs (16).

https://dx.doi.org/10.21037/jmai-2025-123

For learning the initial imputer, we implement the same architecture as used in the MTSIT framework (18), which consists of a Transformer encoder and linear decoder. We implement a Transformer encoder with three encoder blocks and eight heads, with the dimensionality of the model set to 64 and dimensionality of the feed-forward layer set to 256, following similar configurations from prior work (18,55). Each model is trained for 6K epochs using the RAdam optimizer (60) with the learning rate set to 1×10^{-3} with early checkpointing. We found these training parameters to be sufficient to yield solutions with imputation performance (i.e., given by nRMSD) comparable to results from the literature (22). In addition, we sampled initial imputers over a range of efficiencies spanning 10% to 95%, sampled every 5% to cover different levels of desired efficiency (i.e., eff_{range}).

For generating diverse PMD-imputer pairs, we sweep over various hyperparameter configurations defined by λ_{mw} and η with grid-based search. We set $\lambda_{mw} = \{1\times10^{-7},1\times10^{-6},1\times10^{-5}\}$ and $\eta = \{0.1,0.5,1,5,10\}$ since these were sufficient to generate new solutions (i.e., ones with better properties as measured on the validation/tuning set) under METRIK. We adopt the same architectural and training parameters used for learning the initial imputer although we use the last checkpoint since the PMD weights stabilize towards the end of training.

For PMD-imputer selection, we calculate two-sided 95% confidence intervals using the bootstrap algorithm (23) and set the number of bootstraps to 100.

Computing requirements

Running the METRIK framework to generate PMD-imputer pairs requires at most 49 CPU core hours:

- training the initial imputers requires up to 3 CPU core hours, as each imputer takes at most 10 minutes to train,
- generating candidate PMD-imputer pairs takes up to 45 CPU core hours; for each initial imputer, we sweep over the mask weight and learning rate parameters, with each model taking at most 10 minutes to train,
- selecting the final PMD-imputer pair takes no more than 10 minutes on a single CPU core.

Each step within METRIK requires at most 4GB of memory on the sampled datasets (more could be required depending on the size of the data matrix).

We implement all experiments using standard numerical Python packages and PyTorch (61). Since the MTSIT algorithm is based on the model from (55), the code is obtained from (62) and is distributed under the MIT license; the copyright is under George Zerveas since Year 2021. In addition, the code for the mask learning algorithm is obtained from (63) and is distributed under BSD 3-Clause "New" or "Revised" License; the copyright is under Robert Csordas since Year 2020.

Appendix 3 Datasets

Details of the data source and our process for obtaining a convenience sample from the dataset are provided next.

Source

All datasets can be obtained from NINDS (24) by filling out the data request form (https://www.ninds.nih.gov/sites/default/files/migrate-documents/sig_form_revised_508c.pdf). The terms of use for the datasets require that the dataset be used for research purposes, that results from the research be published, that the dataset not be shared without permission from NINDS, and that NINDS, along with the study publication and trial investigators associated with the dataset, be acknowledged in the publication.

Convenience Sample Generation

Per dataset, we obtain a convenience sample from the raw clinical dataset by applying a sequence of steps. First, we remove any data collection forms that are not prescribed according to the trial protocol, comprised measurements that have no associated study visit, or are inconveniently formatted. We also remove any metrics that are derived from other metrics. Then, we remove entries including duplicate rows, NaN indices, duplicate columns or metrics, subjects not assigned to any treatment group in the RCT, and timepoints or visit ids not part of the trial protocol. Next, we post-process the data by standardizing NaN placeholders, removing metrics with mixed data types (e.g., string and numerical), and removing metrics that are constant or entirely NaN. We automatically determine the type of each metric (i.e., continuous *vs.* categorical) based on the data type and range of values, and also drop metrics that have very low variance to avoid issues with normalization. We also automatically determine p_c, the complete data collection protocol, by identifying measurements (i.e., specific metric-timepoint pairs) with sufficiently low missingness rates. Subject masks S are automatically

https://dx.doi.org/10.21037/jmai-2025-123

determined based on NaN entries. We also obtain the treatment assignment per trial participant and the list of timepoints considered part of baseline data collection.

References

51. Luo Y, Szolovits P, Dighe AS, et al. 3D-MICE: integration of cross-sectional and longitudinal imputation for multi-analyte longitudinal clinical data. J Am Med Inform Assoc 2018;25:645-53.
52. Xu X, Liu X, Kang Y, et al. A Multi-directional Approach for Missing Value Estimation in Multivariate Time Series Clinical Data. J Healthc Inform Res 2020;4:365-82.
53. Du T, Melis L, Wang T. ReMasker: Imputing Tabular Data with Masked Autoencoding. arXiv 2023. arXiv:2309.13793v1.
54. Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. In: Proceedings of the Annual Conference on Neural Information Processing Systems; 2017.
55. Zerveas G, Jayaraman S, Patel D, et al. A transformer-based framework for multivariate time series representation learning. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD); 2021:2114-24.
56. Santoni ML, Raponi E, Leone RD, et al. Comparison of high-dimensional Bayesian optimization algorithms on BBOB. ACM Trans Evol Learn Optim 2024;4:133.
57. Han E, Arora I, Scarlett J. High-dimensional Bayesian optimization via tree-structured additive models. In: Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI); 2021:7630-8.
58. Jedrzejewski-Szmek Z, Abrahao KP, Jedrzejewska-Szmek J, et al. Parameter optimization using covariance matrix adaptation—evolutionary strategy (CMA-ES), an approach to investigate differences in channel properties between neuron subtypes. Front Neuroinform 2018;12:47.
59. Jang E, Gu S, Poole B. Categorical reparameterization with Gumbel-Softmax. arXiv 2017. arXiv:1611.01144v5.
60. Liu L, Jiang H, He P, et al. On the variance of the adaptive learning rate and beyond. arXiv 2019. arXiv:1908.03265v4.
61. Paszke A, Gross S, Massa F, et al. PyTorch: An imperative style, high-performance deep learning library. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems (NeurIPS); 2019.
62. Zerveas G. Multivariate time series transformer framework. 2021. Available online: https://github.com/gzerveas/mvts_transformer
63. Csordás R. Codebase for inspecting modularity of neural networks. 2021. Available online: https://github.com/RobertCsordas/modules

https://dx.doi.org/10.21037/jmai-2025-123