N^2: A Unified Python Package and Test Bench for Nearest Neighbor-Based Matrix Completion

Caleb Chin[*] Cornell University
Aashish Khubchandani Cornell University
Harshvardhan Maskara Cornell University
Kyuseong Choi Cornell University
Jacob Feitelberg Columbia University
Albert Gong Cornell University
Manit Paul University of Pennsylvania
Tathagata Sadhukhan Cornell University
Anish Agarwal Columbia University
Raaz Dwivedi Cornell University

Abstract

Nearest neighbor (NN) methods have re-emerged as competitive tools for matrix completion, offering strong empirical performance and recent theoretical guarantees, including entry-wise error bounds, confidence intervals, and minimax optimality. Despite their simplicity, recent work has shown that NN approaches are robust to a range of missingness patterns and effective across diverse applications. This paper introduces $\mathbf{N}^2$, a unified Python package and testbed that consolidates a broad class of NN-based methods through a modular, extensible interface. Built for both researchers and practitioners, $\mathbf{N}^2$ supports rapid experimentation and benchmarking. Using this framework, we introduce a new NN variant that achieves state-of-the-art results in several settings. We also release a benchmark suite of real-world datasets—from healthcare and recommender systems to causal inference and LLM evaluation—designed to stress-test matrix completion methods beyond synthetic scenarios. Our experiments demonstrate that while classical methods excel on idealized data, NN-based techniques consistently outperform them in real-world settings.

1 Introduction

Nearest neighbor methods are a class of non-parametric algorithms widely used for regression, classification and pattern recognition. Due to their scalability and success under models with minimal assumptions, nearest neighbor methods have recently been adopted for practical fields such as matrix completion and counterfactual inference in panel data settings. Matrix completion is a well-established field that supplies practitioners with many tools to recover underlying matrices using partial or even noisy observations [HMLZ15, Cha15, KMO10], with recommendation systems [KBV09, Rec11] as an important use-case. Panel data counterfactual inference aims at learning the treatment effect of policies across time [Bai09, BN21, ABD$^+$21]. One important example is individualized healthcare predictions [KSS$^+$19]. Nearest neighbor methods were recently recognized as effective in providing granular inference guarantees for both matrix completion and counterfactual inference when either the missingness or the policy treatment are not completely random and confounded [MC19, DTT$^+$22a, ADSS23].

Despite nearest neighbor methods popularity, there is no unified package that lets a user easily switch between different kinds of nearest neighbor algorithms for matrix completion and counterfactual

[*]ctc92@cornell.edu

AI for Tabular Data workshop at EurIPS 2025.

inference. In this paper, we present a package[2] to unify several nearest neighbor methods under a single interface, so users can easily choose the method that suits their data the best.

1.1 Our contributions

Overall, our contributions in this paper are summarized below:

1. We provide a unified, easy to implement nearest neighbor library that contains a breadth of nearest neighbor algorithms for matrix completion problems.
2. We present a unified framework for nearest neighbor algorithms that facilitates extending to new variants.
3. We demonstrate our library's wide applicability through several real-world data sets in a new test bench called $\mathbf{N}^2$**-Bench**.

Existing software for matrix completion and nearest neighbors Scikit-Learn [PVG+11], a popular Python package for machine learning tools, implements a simple k-nearest neighbor algorithm for imputing missing values in a feature matrix. However, their implementation is designed for the feature matrix setting. So, neighbors are only defined across samples (row-wise). Additionally, they do not provide any implementation for more advanced nearest neighbor algorithms, nor does their package allow for easy extendability like our proposed package.

2 Nearest Neighbors for Matrix Completion

We now introduce the mathematical model for matrix completion:

$$\text{for} \quad i \in [N], t \in [T]: \\ Z_{i,t} := \begin{cases} X_1(i,t), ..., X_n(i,t) \sim \mu_{i,t} & \text{if} \quad A_{i,t} = 1, \\ \text{unknown} & \text{if} \quad A_{i,t} = 0. \end{cases} \tag{1}$$

In other words, for matrix entries where $A_{i,t} = 1$, we observe n measurements $Z_{i,t}$ that takes value $X_{i,t}$ realized from distribution $\mu_{i,t}$. When $n = 1$, i.e., $Z_{i,t} = X_1(i,t)$, we refer to (1) as the *scalar matrix completion* model; scalar matrix completion is the most common problem posed in the literature [CR12, Rec11, KBV09, HMLZ15, Cha15, DTT+22a, DTT+22b, ADSS23], where the goal is to learn the mean of the underlying distributions $\{\theta_{i,t} = \int x d\mu_{i,t}(x)\}_{i\in[N],t\in[T]}$. When there are more than one observed measurements per entry, i.e., $Z_{i,t} = [X_1(i,t), ..., X_n(i,t)]$ for $n \geq 2$, we refer to (1) as the *distributional matrix completion* problem, the goal being the recovery of the distributions as a whole. We refer the readers to App. A for a detailed discussion on the structural assumptions imposed on the model (1).

3 $\mathbf{N}^2$ Package and Interface

We now present our unified Python package, $\mathbf{N}^2$, for nearest neighbor algorithms for matrix completion. In particular, we provide a class structure which abstracts the estimation procedure utilized in each different nearest neighbor method and is facilitated by DISTANCE and AVERAGE modules described in more detail in App. C. On top of that, our library enables easy extension to other nearest neighbors algorithms and other data types on top of scalars and distributions. For example, as long as a distance and average notion are well defined, our library can be easily applied to a matrix of images or text strings.

Interface. The core functionality of $\mathbf{N}^2$ is based on two abstract classes: `EstimationMethod` and `DataType`. The details of these classes can be found in App. B. To use our library, a user simply has to instantiate a composite class `NearestNeighborImputer` with their `EstimationMethod` and `DataType` of choice. We provide constructor functions to automatically create popular `NearestNeighborImputer` classes such as a two-sided nearest neighbor estimator with the scalar data type. From a design pattern point of view, this is known as a *Composite* design pattern [GHJV93,

[2] `https://anonymous.4open.science/r/NearestNeighbors-DAF3`

pg. 163]. We use this design pattern so that anyone looking to customize the estimation procedure can do so for any kind of data type simultaneously. Similarly, with the exception of doubly robust estimators, each estimation procedure works out of the box with any data type that implements the `DataType` abstract class. The Doubly robust estimation method does not work out of the box with distributions because a subtraction operation is not well defined in the distribution space.

Finally, the user simply needs to input (i) a data matrix, (ii) a mask matrix which specifies which values are missing, and (iii) the row and column to impute. Thus, a user can test out different estimation procedures by changing just one line of code. Separately from the core functionality, we have also implemented several cross-validation classes detailed in App. D which take in a `NearestNeighborImputer` class and find the best hyperparameters to use (e.g., distance thresholds and weights).

4 $\mathbf{N}^2$-Bench and Results

In this section, we evaluate several nearest neighbor algorithms provided by our library, $\mathbf{N}^2$, on real-world data. As part of our package, we include data loaders which automatically download the necessary datasets and format them for evaluation. These datasets and loaders comprise our proposed benchmark for nearest neighbor matrix completion algorithms, $\mathbf{N}^2$-**Bench**. We also test several existing popular matrix completion techniques [HMLZ15, Cha15]. For details on our experimental setup, computing hardware, and boxplot generation, see App. E.

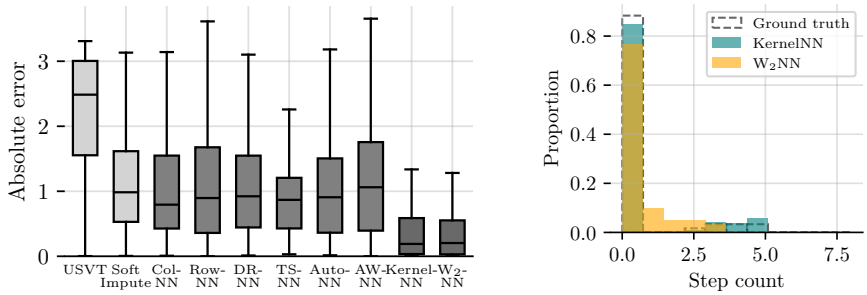
4.1 Personalized healthcare: HeartSteps

The HeartSteps V1 study (HeartSteps study for short) is a clinical trial designed to measure the efficacy of the HeartSteps mobile application for encouraging non-sedentary activity [KSS+19]. The HeartSteps V1 data and its subsequent extensions have been widely used for benchmarking a variety of tasks including counterfactual inference of treatment effect [DTT+22a, CFC+24], reinforcement learning for intervention selection [LGKM20], and micro-randomized trial design [QWC+22]. In the HeartSteps study, $N = 37$ participants were under a 6-week period micro-randomized trial, where they were provided with a mobile application and an activity tracker. Participants independently received a notification with probability $p = 0.6$ for 5 pre-determined decision points per day for 40 days ($T = 200$). We denote observed entries $Z_{i,t}$ as the mean participant step count for one hour after a notification was sent and unobserved entries as the unknown step count for decision points where no notification was sent. Our task is to estimate the counterfactual outcomes: the participant's step count should they have received a different treatment (notification or no notification) than they did at specific time points during the study.

Results & Discussion. We benchmark the performance of the matrix completion methods by measuring absolute error on held-out observed step counts across 10 participants in the last 50 decision points. We use the remaining data to find nearest neighbor hyperparameters using cross-validation. To benchmark distributional nearest neighbors methods (KernelNN and W_2NN) against the scalar methods, we first set each entry to have the number of samples $n = 60$, where each sample is the 1 minute step count before imputation. Then, we take the mean of the imputed empirical distribution as the estimate.

In Fig. 1(a), we compare the absolute error of the imputed values across the nearest neighbor and baseline methods. The scalar nearest neighbor methods far out-perform USVT and are on par with SoftImpute. The two distributional nearest neighbor methods far outperform all methods operating in the scalar setting; it suggests that matching by distributions collect more homogeneous neighbors, thereby decreases the bias of the method, compared to matching only the first moments as done in most scalar matrix nearest neighbor methods.

In Fig. 1 panel (b), we show an example of an imputed entry in the distributional nearest neighbors setting. In this case, the ground truth distribution is bimodal, as the participant was largely sedentary (0 steps) with small amounts of activity. While both KernelNN and W_2NN capture the sedentary behavior of the participant, KernelNN is able to recover the bimodality of the original distribution whereas W_2NN cannot.



(a) Absolute error of mean step count prediction (b) KernelNN vs. W_2NN

Figure 1: **HeartSteps: estimating step count under scalar and distributional matrix completion settings.** Panel (a) shows the absolute error of predicted step count of the nearest neighbor methods against matrix completion baselines (SoftImpute, USVT). Panel (b) shows an example of an imputed entry in the distributional matrix completion setting.

4.2 Movie recommendations: MovieLens

The MovieLens 1M dataset [HK15] contains 1 million ratings (1–5 stars) from 6,040 users on 3,952 movies. Collaborative filtering on MovieLens has long been a benchmark for matrix-completion methods: neighborhood-based algorithms [SKKR01], latent-factor models [KBV09], and, more recently, nearest neighbors interpreted as blind regression under a latent–variable model [LSSY19]. These assist practitioners in data-driven recommendation systems, since more accurate rating imputation directly drives better personalized suggestions and user engagement. This is a standard scalar matrix completion problem with $N = 6,040$ and $T = 3,952$. Each rating is an integer in $\{1, \ldots, 5\}$. The dataset has a very high percentage of missing values: 95.53% missing. Our task is to estimate unobserved ratings using various matrix completion algorithms. We benchmark the performance of nearest neighbors against matrix factorization by measuring absolute error on held-out ratings. See App. E.3 for additional details on the dataset.

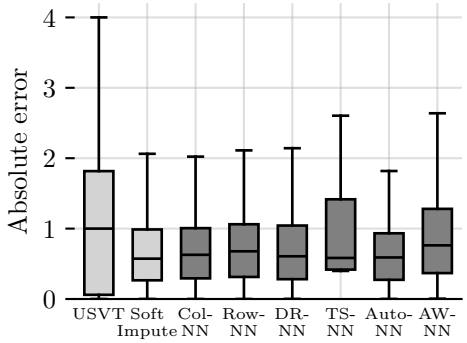


Figure 2: **MovieLens: Estimation error for a random subsample of size 500.** For experimental settings and discussion see Sec. 4.2.

Results & Discussion. We fit the nearest neighbor methods using a random sample of size 100 from the first 80% of the dataset to choose nearest neighbor hyperparameters via cross-validation.We then test the method on a random subsample of size 500 from the last 20% of the dataset. As observed in Fig. 2, all nearest neighbor methods have a lower average error than USVT and a much lower standard deviation of errors, with ColNN, RowNN, DRNN, and AutoNN performing the best out of the nearest neighbor methods. SoftImpute performs on par with the nearest neighbor methods. Note that the nearest neighbor methods perform well even while only being trained on a tiny subset of the data of size 100 out of the 1 million ratings available.

5 Conclusion

In this paper, we present a unified framework, Python library ($\mathbf{N}^2$), and test bench ($\mathbf{N}^2$-**Bench**) for nearest neighbor-based matrix completion algorithms. We demonstrate how our library supports a diverse set of datasets spanning patient-level healthcare causal inference (HeartSteps) and recommen-

dation systems (MovieLens). Our framework and library facilitates researchers and practitioners to try NN methods on novel datasets as well as extend the package with more complex methods.

In future work, we plan on speeding up the runtime of $\mathbf{N}^2$, particularly for commonly used settings such as scalars-valued matrices. We also plan on adding support for distributed datasets too large to fit into memory. Finally, we plan on extending the library to other nearest neighbor algorithms, such as approximate nearest neighbors methods and ones that use linear regression instead of simple averaging.

References

[ABD+21] Susan Athey, Mohsen Bayati, Nikolay Doudchenko, Guido Imbens, and Khashayar Khosravi. Matrix completion methods for causal panel data models. *Journal of the American Statistical Association*, 116(536):1716–1730, 2021.

[ADH10] Alberto Abadie, Alexis Diamond, and Jens Hainmueller. Synthetic control methods for comparative case studies: Estimating the effect of california's tobacco control program. *Journal of the American statistical Association*, 105(490):493–505, 2010.

[ADSS23] Anish Agarwal, Munther Dahleh, Devavrat Shah, and Dennis Shen. Causal matrix completion. In *The Thirty Sixth Annual Conference on Learning Theory*, pages 3821–3826. PMLR, 2023.

[AI22] Susan Athey and Guido W Imbens. Design-based analysis in difference-in-differences settings with staggered adoption. *Journal of Econometrics*, 226(1):62–79, 2022.

[Bai09] Jushan Bai. Panel data models with interactive fixed effects. *Econometrica*, 77(4):1229–1279, 2009.

[BBBK11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[BGKL17] Jérémie Bigot, Raúl Gouet, Thierry Klein, and Alfredo López. Geodesic PCA in the Wasserstein space by convex PCA. *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, 53(1):1 – 26, 2017.

[Big20] Bigot, Jérémie. Statistical data analysis in the wasserstein space*. *ESAIM: ProcS*, 68:1–19, 2020.

[BN21] Jushan Bai and Serena Ng. Matrix completion, counterfactuals, and factor analysis of missing data. *Journal of the American Statistical Association*, 116(536):1746–1763, 2021.

[BYC13] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123. PMLR, 2013.

[CAD20] Samuel Cohen, Michael Arbel, and Marc Peter Deisenroth. Estimating barycenters of measures in high dimensions. *arXiv preprint arXiv:2007.07105*, 2020.

[CFC+24] Kyuseong Choi, Jacob Feitelberg, Caleb Chin, Anish Agarwal, and Raaz Dwivedi. Learning counterfactual distributions via kernel nearest neighbors. *arXiv preprint arXiv:2410.13381*, 2024.

[Cha15] Sourav Chatterjee. Matrix estimation by Universal Singular Value Thresholding. *The Annals of Statistics*, 43(1):177 – 214, 2015.

[CR12] Emmanuel Candes and Benjamin Recht. Exact matrix completion via convex optimization. *Communications of the ACM*, 55(6):111–119, 2012.

[DTT+22a] Raaz Dwivedi, Katherine Tian, Sabina Tomkins, Predrag Klasnja, Susan Murphy, and Devavrat Shah. Counterfactual inference for sequential experiments. *arXiv preprint arXiv:2202.06891*, 2022.

[DTT+22b] Raaz Dwivedi, Katherine Tian, Sabina Tomkins, Predrag Klasnja, Susan Murphy, and Devavrat Shah. Doubly robust nearest neighbors in factor models. *arXiv preprint arXiv:2211.14297*, 2022.

[DZCM22] Raaz Dwivedi, Kelly Zhang, Prasidh Chhabaria, and Susan Murphy. Deep dive into personalization. *Working paper*, 2022.

[FCAD24] Jacob Feitelberg, Kyuseong Choi, Anish Agarwal, and Raaz Dwivedi. Distributional matrix completion via nearest neighbors in the wasserstein space. *arXiv preprint arXiv:2410.13112*, 2024.

[GHC+25] Jadon Geathers, Yann Hicke, Colleen Chan, Niroop Rajashekar, Justin Sewell, Susannah Cornes, Rene Kizilcec, and Dennis Shung. Benchmarking generative ai for scoring medical student interviews in objective structured clinical examinations (osces). *arXiv preprint arXiv:2501.13957*, 2025.

[GHJV93] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design patterns: Abstraction and reuse of object-oriented design. In *ECOOP'93—Object-Oriented Programming: 7th European Conference Kaiserslautern, Germany, July 26–30, 1993 Proceedings 7*, pages 406–431. Springer, 1993.

[HBB+20] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[HK15] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), December 2015.

[HMLZ15] 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.

[Hun07] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[KBV09] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[KMO10] Raghunandan H Keshavan, Andrea Montanari, and Sewoong Oh. Matrix completion from a few entries. *IEEE transactions on information theory*, 56(6):2980–2998, 2010.

[KSS+19] Predrag Klasnja, Shawna Smith, Nicholas J Seewald, Andy Lee, Kelly Hall, Brook Luers, Eric B Hekler, and Susan A Murphy. Efficacy of contextually tailored suggestions for physical activity: a micro-randomized optimization trial of heartsteps. *Annals of Behavioral Medicine*, 53(6):573–582, 2019.

[LGKM20] Peng Liao, Kristjan Greenewald, Predrag Klasnja, and Susan Murphy. Personalized heartsteps: A reinforcement learning algorithm for optimizing physical activity. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 4(1), March 2020.

[LR19] Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*, volume 793. John Wiley & Sons, 2019.

[LSSY19] Yihua Li, Devavrat Shah, Dogyoon Song, and Christina Lee Yu. Nearest neighbors for matrix estimation interpreted as blind regression for latent variable model. *IEEE Transactions on Information Theory*, 66(3):1760–1784, 2019.

[MC19] Wei Ma and George H Chen. Missing not at random in matrix completion: The effectiveness of estimating missingness probabilities under a low nuclear norm assumption. *Advances in neural information processing systems*, 32, 2019.

[MFS+17] Krikamol Muandet, Kenji Fukumizu, Bharath Sriperumbudur, Bernhard Schölkopf, et al. Kernel mean embedding of distributions: A review and beyond. *Foundations and Trends® in Machine Learning*, 10(1-2):1–141, 2017.

[OW23] Orzechowski and Walker. The Tax Burden on Tobacco, 1970-2019 | Data | Centers for Disease Control and Prevention — data.cdc.gov. `https://data.cdc.gov/api/views/7nwe-3aj9/rows.csv?accessType=DOWNLOAD`, 2023. [Accessed 16-05-2025].

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[PXW+24] Felipe Maia Polo, Ronald Xu, Lucas Weber, Mírian Silva, Onkar Bhardwaj, Leshem Choshen, Allysson Flavio Melo de Oliveira, Yuekai Sun, and Mikhail Yurochkin. Efficient multi-prompt evaluation of llms. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[QWC+22] Tianchen Qian, Ashley E Walton, Linda M Collins, Predrag Klasnja, Stephanie T Lanza, Inbal Nahum-Shani, Mashfiqui Rabbi, Michael A Russell, Maureen A Walton, Hyesun Yoo, et al. The microrandomized trial for developing digital interventions: Experimental design and data analysis considerations. *Psychological methods*, 27(5):874, 2022.

[Rec11] Benjamin Recht. A simpler approach to matrix completion. *Journal of Machine Learning Research*, 12(12), 2011.

[SKKR01] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, page 285–295, New York, NY, USA, 2001. Association for Computing Machinery.

[SPD24] Tathagata Sadhukhan, Manit Paul, and Raaz Dwivedi. On adaptivity and minimax optimality of two-sided nearest neighbors. *arXiv preprint arXiv:2411.12965*, 2024.

[SPD25] Tathagata Sadhukhan, Manit Paul, and Raaz Dwivedi. Adaptively-weighted nearest neighbors for matrix completion. *arXiv preprint arXiv:2505.09612*, 2025.

A Structural assumptions

Provable guarantees of nearest neighbors in matrix settings (1) can be shown when structural assumptions are imposed on the distributions $\mu_{i,t}$ and the missingness $A_{i,t}$. We collect existing results from [LSSY19, DTT$^+$22b, CFC$^+$24, FCAD24, SPD24, SPD25]. Given data with missing observations from (1), the practitioner is interested in learning information of the distributions, e.g., mean of the distributions $\{\theta_{i,t} = \int x d\mu_{i,t}(x)\}$.

The first assumption specifies the factor structure on the mean; that is, there exists latent factors u_i, v_t that collectively characterize the signal of each entry (i,t) of the matrix [LSSY19, DTT$^+$22a, ADSS23, CFC$^+$24, FCAD24]. Such a factor model is analogous to the low rank assumptions commonly imposed in matrix completion [CR12]. The second assumption specifies how the missing pattern $A_{i,t}$ was generated; for instance missing completely at random (MCAR) assumes that $A_{i,t}$ are independent to all other randomness present in the model (1) and that all entries have positive probability of being observed.

A.1 Factor model

For the scalar matrix completion problem, i.e., (1) with $n = 1$, the main goal is to learn (or impute) the mean of the underlying distribution $\theta_{i,t}$ for any missing entries [LSSY19, DTT$^+$22b, DTT$^+$22a, ADSS23, SPD24, SPD25]. The majority of this literature assumes (i) an additive noise model $\mu_{i,t} = \theta_{i,t} + \varepsilon_{i,t}$ for centered i.i.d. sub-Gaussian noise ε and (ii) mean factor model, i.e., $\theta_{i,t} = f(u_i, v_t)$ for some latent factors u_i, v_t and real valued function f.

For the distributional matrix completion problem (i.e., (1) with $n > 1$) the main goal is to learn the underlying distribution itself [CFC$^+$24, FCAD24]; a factor model is imposed on the distribution as a whole. For instance, a factor model is assumed on the kernel mean embedding of distributions; that is, there exist latent factors u_i and v_t and an operator g such that $\int \mathbf{k}(x, \cdot) d\mu_{i,t}(x) = g(u_i, v_t)$.

A.2 Missingness pattern

For both the scalar and distributional matrix completion problem (1), the missing pattern (i.e., how the missingness $A_{j,s}$ was generated) can be categorized into three classes using the taxonomy of [LR19]: missing-completely-at-random (MCAR), missing-at-random (MAR) and missing-not-at-random (MNAR). MCAR assumes that the missingness $A_{i,t}$ is exogenous (independently generated from all the randomness in the model) and i.i.d. with propensity $\mathbb{P}(A_{i,t} = 1) = p > 0$ for all (i,t). MAR is a more challenging scenario compared to MCAR as missingness is not exogenous, but its randomness depends on the observations. Further, propensities $p_{i,t}$ may differ for entries (i,t) but positivity still holds, i.e., $\min_{i\in[N],t\in[T]} p_{i,t} > 0$. An important instance for MAR is the adaptive randomized policies [DZCM22]. The MNAR setup is the most challenging as it assumes the missingness depends on the unobserved latent confounders, while positivity may also be violated, i.e., $\min_{i\in[N],t\in[T]} p_{i,t} = 0$. The staggered adoption pattern, where a unit remains treated once a unit is treated at some adoption time, is a popular example of MNAR, mainly because positivity is violated. See [ABD$^+$21, AI22] for more details on staggered adoption.

We briefly outline the structural assumptions existing nearest neighbor methods were shown to work with provable guarantees; for all the existing methods, factor models (with slightly different details; compare the mean factorization [LSSY19] and the distribution factorization [CFC$^+$24, FCAD24]) are all commonly assumed.

- (Scalar matrix completion) The vanilla versions of nearest neighbors (RowNN) in [LSSY19, DTT$^+$22a] are shown to work for MCAR and MAR setup; the latter shows that simple nearest neighbors can provably impute the mean when the missingness is fully adaptive across all users and history. The variants of vanilla nearest neighbors DRNN [DTT$^+$22b] is proven to work under MCAR, while TSNN [SPD24] is proven to work under unobserved confounding, i.e., MNAR.
- (Distributional matrix completion) The KernelNN [CFC$^+$24] is shown to recover the underlying distribution under MNAR, whereas W_2NN [FCAD24] is shown to work under MCAR.

B Class Structure Details

The core functionality of $\mathbf{N}^2$ is based on two abstract classes: `EstimationMethod` and `DataType`.

`EstimationMethod` classes contain the logic to impute a missing entry such as how to use calculated distances. We separate this from the `DataType` abstraction because several estimation methods can be used for multiple data types. For example, `RowRowEstimator` implements the RowNN procedure for any data type given to it, such as scalars or distributions.

`DataType` classes implement a *distance* and *average* function for any kind of data type. For scalars we use squared distance and simple averaging. For distributions, we implement two metrics, Wasserstein ($\mathsf{W}_2\mathsf{NN}$) and kernel maximum mean discrepancy (MMD, KernelNN). This abstract class allows for our package to extend to any data types beyond the ones we tested. For instance, a practitioner can easily add a `DataType` for text strings which uses vector embeddings to find distances and averages between between strings without needing to rewrite any of the estimation procedure.

C Nearest neighbor algorithms

C.1 Unified framework

We introduce two general modules (namely DISTANCE and AVERAGE) from which the variants of nearest neighbors are constructed. We introduce several shorthands used in the modules. Denote the collection of measurements, missingness, and weights:

$$\mathcal{Z} := \left\{Z_{j,s}\right\}_{j\in[N],s\in[T]}, \quad \mathcal{A} := \{A_{j,s}\}_{j\in[N],s\in[T]},$$
$$\text{and} \quad \mathcal{W} := \{w_{j,s}\}_{j\in[N],s\in[T]}.$$

Let $\varphi(x,x')$ be a metric between $x,x' \in \mathcal{X}$ for some space $\mathcal{X}$. Further define $\widehat{\varphi}(Z_{i,t},Z_{j,s})$ as a data-dependent distance between any two observed entries (i,t) and (j,s) of the matrix (1). The two modules can now be defined:

(i) DISTANCE$(\widehat{\varphi},\mathcal{Z},\mathcal{A})$: Additional input is the data-dependent distance between entries of matrix $\widehat{\varphi}$ and output is the collection of row-wise and column-wise distance of matrix:

$$\rho^{\text{row}}_{i,j} := \frac{\sum_{s\neq t} A_{i,s}A_{j,s}\widehat{\varphi}(Z_{i,s},Z_{j,s})}{\sum_{s\neq t} A_{i,s}A_{j,s}} \quad \text{and}$$
$$\rho^{\text{col}}_{t,s} := \frac{\sum_{j\neq i} A_{j,t}A_{j,s}\widehat{\varphi}(Z_{j,t},Z_{j,s})}{\sum_{j\neq i} A_{j,t}A_{j,s}},$$

(ii) AVERAGE$(\varphi,\mathcal{W},\mathcal{Z},\mathcal{A})$: Additional input are the weights $\mathcal{W}$, metric φ and output is the optimizer

$$\widehat{\theta} = \operatorname*{argmin}_{x\in\mathcal{X}} \sum_{j\in[N],s\in[T]} w_{j,s}A_{j,s}\varphi(x,Z_{j,s}).$$

The DISTANCE module calculates the row-wise and column-wise distance of the matrix, by taking the average of the observed entry-wise distance $\widehat{\varphi}(\cdot,\cdot)$. The AVERAGE module calculates the weighted average of observed measurements, where the notion of average depends on the metric φ and the space $\mathcal{X}$ on which the metric φ is defined. Notably, the weights $\mathcal{W}$ in the AVERAGE module encodes the entry information of the estimand.

Remark 1 *The vanilla row-wise nearest neighbors [LSSY19] that targets the mean $\theta_{i,t} = \int x d\mu_{i,t}(x)$ of entry (i,t) is recovered by first applying* DISTANCE *with $\widehat{\varphi}(Z_{j,s},Z_{j',s'}) = (Z_{j,s}-Z_{j',s'})^2$, applying* AVERAGE *with the non-smooth weight $w_{j,s} = \mathbf{1}(\rho^{\text{row}}_{i,j}\leq\eta_1)\cdot\mathbf{1}(\rho^{\text{col}}_{s,t}\leq 0)$, and using the metric $\varphi(x,y) = (x-y)^2$. Note that the non-smooth weight satisfies $w_{j,t} = \mathbf{1}(\rho^{\text{row}}_{i,j}\leq\eta_1)$, whereas $w_{j,s} = 0$ for $s\neq t$; by defining the nearest neighbor set $\mathbf{N}_{t,\eta_1} := \{j\in[N] : \rho^{\text{row}}_{j,t}\leq\eta_1\}$, the* AVERAGE *module output can be rewritten as* $\operatorname{argmin}_{x\in\mathbb{R}}\sum_{j\in\mathbf{N}_{t,\eta_1}} A_{j,t}(x-Z_{j,t})^2 = |\mathbf{N}_{t,\eta_1}|^{-1}\sum_{j\in\mathbf{N}_{t,\eta_1}} Z_{j,t}$.

C.2 Existing methods

We present existing variants of nearest neighbors using the two modules introduced App. C.1; all the methods presented here are recovered by sequentially applying DISTANCE and AVERAGE with the appropriate specification of $\widehat{\varphi}$, φ and $\mathcal{W}$.

For simple notation, we introduce a shorthand for the non-continuous weight

$$w_{j,s}^{(i,t)}(\eta_1, \eta_2) := \mathbf{1}(\rho_{i,j}^{\mathrm{row}} \leq \eta_1) \cdot \mathbf{1}(\rho_{s,t}^{\mathrm{col}} \leq \eta_2).$$

All methods except AWNN and our newly proposed AutoNN, have binary weights i.e., $w_{j,s} \in \{0, 1\}$. AutoNN, detailed in App. C.7, uses weights to carefully pool together the benefits of TSNN and DRNN. AWNN [SPD25] improves upon RowNN by adaptively choosing the weights which optimally balances the bias-variance tradeoff of RowNN as follows

$$\begin{aligned} &\big(w_1^\star(i,t), ..., w_N^\star(i,t)\big) := \\ &\underset{(v_1,\ldots,v_N)\in\Delta_N}{\operatorname{argmin}} \; 2\log(2N)\widehat{\sigma}^2 \sum_{k\in[N]} v_k^2 + \sum_{k\in[N]} v_k A_{k,t} \rho_{i,k}^{\mathrm{row}}. \end{aligned} \tag{2}$$

where $\widehat{\sigma}^2$ is the estimated error and Δ_N is a simplex in $\mathbb{R}^N$; see [SPD25] for details of (2). Tab. 1 contains a concise summary of the existing nearest neighbor variants; see App. C for a detailed exposition for each methods.

Table 1: **Variants of nearest neighbors for matrix completion.**

Type	Method	$\widehat{\varphi}(x,y)$	$\varphi(x,y)$	$w_{j,s}$
$n=1$	RowNN [LSSY19] (Alg. 1)	$(x-y)^2$	$(x-y)^2$	$\mathbf{1}(\rho_{i,j}^{\mathrm{row}} \leq \eta_1, \rho_{s,t}^{\mathrm{col}} \leq 0)$
	ColNN [LSSY19] (Alg. 1)	$(x-y)^2$	$(x-y)^2$	$\mathbf{1}(\rho_{i,j}^{\mathrm{row}} \leq 0, \rho_{s,t}^{\mathrm{col}} \leq \eta_2)$
	TSNN [SPD24] (Alg. 2)	$(x-y)^2$	$(x-y)^2$	$\mathbf{1}(\rho_{i,j}^{\mathrm{row}} \leq \eta_1, \rho_{s,t}^{\mathrm{col}} \leq \eta_2)$
	AWNN [SPD25] (Alg. 5)	$(x-y)^2$	$(x-y)^2$	$w_j^\star(i,t) \cdot \mathbf{1}(\rho_{s,t}^{\mathrm{col}} \leq 0)$
	DRNN [DTT+22b] (Alg. 3)	RowNN + ColNN − TSNN		
	AutoNN (App. C.7)	$\alpha \cdot$ DRNN $+ (1-\alpha) \cdot$ TSNN		
$n>1$	KernelNN [CFC+24] (Alg. 4)	$\widehat{\mathsf{MMD}}_{\mathbf{k}}^2(x,y)$	$\mathsf{MMD}_{\mathbf{k}}^2(x,y)$	$\mathbf{1}(\rho_{i,j}^{\mathrm{row}} \leq \eta_1, \rho_{s,t}^{\mathrm{col}} \leq 0)$
	W_2NN [FCAD24] (Alg. 4)	$\widehat{\mathsf{W}}_2^2(x,y)$	$\mathsf{W}_2^2(x,y)$	$\mathbf{1}(\rho_{i,j}^{\mathrm{row}} \leq \eta_1, \rho_{s,t}^{\mathrm{col}} \leq 0)$

Under the distributional matrix completion setting ($n > 1$ in (1)), the methods KernelNN and W_2NN in Tab. 1 take $\mu, \nu \in \mathcal{X}$ as square integrable probability measures, and $\varphi(\mu, \nu)$ as either the squared maximum mean discrepency (i.e. $\mathsf{MMD}_{\mathbf{k}}^2(\mu, \nu)$, see [MFS+17]) or squared Wasserstein metric (i.e., $\mathsf{W}_2(\mu, \nu)$, see [Big20]). Further, the entry-wise distance $\widehat{\varphi}(x, y)$ in this case is either the unbiased U-statistics estimator $\widehat{\mathsf{MMD}}_{\mathbf{k}}^2(Z_{i,t}, Z_{j,s})$ for $\mathsf{MMD}_{\mathbf{k}}^2(\mu_{i,t}, \mu_{j,s})$ (see [MFS+17]) or the quantile based estimator $\widehat{\mathsf{W}}_2(Z_{i,t}, Z_{j,s})$ for $\mathsf{W}_2(\mu_{i,t}, \mu_{j,s})$ (see [Big20]).

The nearest neighbor methods introduced in Tab. 1 are elaborated in this section. We present two versions of each method; the first version explicitly constructs neighborhoods instead of subtly embedding them in the weights $\mathcal{W}$ of the AVERAGE module, and the second version specifies how each methods can be recovered by applying the two modules, DISTANCE and AVERAGE, sequentially.

C.3 Vanilla nearest neighbors

We elaborate on the discussion in Rem. 1 and provide here a detailed algorithm based on the explicit construction of neighborhoods, which is essentially equivalent to RowNN in Tab. 1. The inputs are measurements $\mathcal{Z}$, missingness $\mathcal{A}$, the target index (i, t), and the radius η.

Step 1: (Distance between rows) Calculate the distance between row i and any row $j \in [N] \setminus \{i\}$ by averaging the squared Euclidean distance across overlapping columns:

$$\rho_{i,j} := \frac{\sum_{s \neq t} A_{i,s} A_{j,s} (Z_{i,s} - Z_{j,s})^2}{\sum_{s \neq t} A_{i,s} A_{j,s}}.$$

Step 2: (Construct neighborhood) Construct a neighborhood of radius η within the tth column using the distances $\{\rho_{i,j} : j \neq i\}$:

$$\mathbf{N}_{t,\eta} := \big\{ j \in [N] \setminus \{i\} : \rho_{i,j} \leq \eta \big\}$$

Step 3: (Average across observed neighbors) Take the average of measurements within the neighborhood:

$$\widehat{\theta}_{i,t,\eta} := \frac{1}{|\mathbf{N}_{t,\eta}|} \sum_{j \in \mathbf{N}_{t,\eta}} A_{j,t} Z_{j,t}.$$

In practice, the input η for RowNN should be optimized via cross-validation; we refer the reader to App. D for a detailed implementation.

We specify the exact implementation of the two modules DISTANCE, AVERAGE to recover RowNN:

Algorithm 1: RowNN for scalar nearest neighbor

Input: $\mathcal{Z}, \mathcal{A}, \eta, (i,t)$

Initialize entry-wise metric $\widehat{\varphi}(Z_{j,s}, Z_{j',s'}) \leftarrow (Z_{j,s} - Z_{j',s'})^2$ and metric $\varphi(x,y) \leftarrow (x-y)^2$

Initialize hyper-parameter $\eta \leftarrow (\eta_1, 0)$

Calculate row-wise metric $\big\{\rho^{\text{row}}_{i,j} : j \neq i\big\} \leftarrow$ DISTANCE$(\widehat{\varphi}, \mathcal{Z}, \mathcal{A})$

Initialize weight $w_{j,s} \leftarrow \mathbf{1}(\rho^{\text{row}}_{i,j} \leq \eta_1, \rho^{\text{col}}_{s,t} \leq \eta_2)$

Calculate average $\widehat{\theta}_{i,t} \leftarrow$ AVERAGE$(\varphi, \mathcal{W}, Z, A)$

return $\widehat{\theta}_{i,t}$

The discussion for RowNN here can be identically made for ColNN as well.

C.4 Two-sided and doubly-robust nearest neighbors

We elaborate on the variants of the vanilla nearest neighbors algorithm TSNN and DRNN in Tab. 1; we first elaborate on an equivalent version of each of the methods which explicitly constructs neighborhoods.

In the following three step procedure, DRNN and TSNN differs in the last averaging step: the inputs are the measurements $\mathcal{Z}$, missingness $\mathcal{A}$, the target index (i,t), and the radii $\eta = (\eta_1, \eta_2)$.

Step 1: (Distance between rows) Calculate the distance between row i and any row $j \in [N] \setminus \{i\}$ and the distance between column t and any column $s \in [T] \setminus \{t\}$:

$$\rho^{\text{row}}_{i,j} := \frac{\sum_{s \neq t} A_{i,s} A_{j,s} (Z_{i,s} - Z_{j,s})^2}{\sum_{s \neq t} A_{i,s} A_{j,s}} \quad \text{and} \quad \rho^{\text{col}}_{t,s} := \frac{\sum_{j \neq i} A_{j,t} A_{j,s} (Z_{j,t} - Z_{j,s})^2}{\sum_{j \neq i} A_{j,t} A_{j,s}}$$

Step 2: (Construct neighborhood) Construct a row-wise and column-wise neighborhood of radius η_1 and η_2 respectively,

$$\mathbf{N}^{\text{row}}_{t,\eta_1} := \big\{ j \in [N] \setminus \{i\} : \rho^{\text{row}}_{i,j} \leq \eta \big\} \quad \text{and} \quad \mathbf{N}^{\text{col}}_{i,\eta_2} := \big\{ s \in [T] \setminus \{t\} : \rho^{\text{col}}_{t,s} \leq \eta \big\}$$

Step 3: (Average across observed neighbors) Take the average of measurements within the neighborhood; the first and the second averaging correspond to DRNN and TSNN respectively:

$$\widehat{\theta}^{\mathsf{DR}}_{i,t,\eta} := \frac{\sum_{j \in \mathbf{N}^{\text{row}}_{t,\eta_1}, s \in \mathbf{N}^{\text{col}}_{i,\eta_2}} A_{j,t} A_{i,s} A_{j,s} \big(Z_{j,t} + Z_{i,s} - Z_{j,s} \big)}{\sum_{j \in \mathbf{N}^{\text{row}}_{t,\eta_1}, s \in \mathbf{N}^{\text{col}}_{i,\eta_2}} A_{j,t} A_{i,s} A_{j,s}} \quad \text{and}$$

$$\widehat{\theta}^{\mathsf{TS}}_{i,t,\eta} := \frac{\sum_{j \in \mathbf{N}^{\text{row}}_{t,\eta_1}, s \in \mathbf{N}^{\text{col}}_{i,\eta_2}} A_{j,s} Z_{j,s}}{\sum_{j \in \mathbf{N}^{\text{row}}_{t,\eta_1}, s \in \mathbf{N}^{\text{col}}_{i,\eta_2}} A_{j,s}}.$$

Next, we specity the exact implemention of the two modules DISTANCE and AVERAGE to recover TSNN and DRNN:

Algorithm 2: TSNN for scalar matrix completion

Input: $\mathcal{Z}, \mathcal{A}, \eta, (i,t)$

Initialize entry-wise metric $\widehat{\varphi}(Z_{j,s}, Z_{j',s'}) \leftarrow (Z_{j,s} - Z_{j',s'})^2$ and metric $\varphi(x,y) \leftarrow (x-y)^2$

Initialize tuning parameter $\eta \leftarrow (\eta_1, \eta_2)$

Calculate row-wise and column-wise metric $\{\rho^{\text{row}}_{i,j} : j \neq i\}, \{\rho^{\text{col}}_{t,s} : s \neq t\} \leftarrow \text{DISTANCE}(\widehat{\varphi}, Z, A)$

Initialize weight $w_{j,s} \leftarrow \mathbf{1}(\rho^{\text{row}}_{i,j} \leq \eta_1, \rho^{\text{col}}_{s,t} \leq \eta_2)$

Calculate average $\widehat{\theta}_{i,t} \leftarrow \text{AVERAGE}(\varphi, \mathcal{W}, \mathcal{Z}, \mathcal{A})$

return $\widehat{\theta}_{i,t}$

For DRNN algorithm below, we consider $\mathcal{Z}$ and $\mathcal{A}$ to be $N \times T$ sized matrices, so that their transpose is well defined. Then note that ColNN is simply applying Alg. 1 with transposed observation matrices.

Algorithm 3: DRNN for scalar matrix completion

Input: $\mathcal{Z}, \mathcal{A}, \eta, (i,t)$

Initialize RowNN $\leftarrow$ Alg. 1 with inputs $(\mathcal{Z}, \mathcal{A}, \eta, (i,t))$ and $\eta \leftarrow (\eta_1, 0)$

Initialize ColNN $\leftarrow$ Alg. 1 with input $(\mathcal{Z}^T, \mathcal{A}^T, \eta, (i,t))$ and $\eta \leftarrow (\eta_1, 0)$

Initialize TSNN $\leftarrow$ Alg. 2 with inputs $(\mathcal{Z}, \mathcal{A}, \eta, (i,t))$ and $\eta \leftarrow (\eta_1, \eta_2)$

Calculate $\widehat{\theta}_{i,t} \leftarrow \text{RowNN} + \text{ColNN} - \text{TSNN}$

return $\widehat{\theta}_{i,t}$

C.5 Distributional nearest neighbors

Unlike the scalar nearest neighbor methods, distributional nearest neighbors necessitate a distributional notion of distance between rows and columns of matrix and a distributional analog of averaging. [CFC$^+$24] and [FCAD24] use maximum mean discrepency (in short MMD) of kernel mean embeddings [MFS$^+$17] and Wasserstein metric (in short W_2) [Big20] respectively both for defining the distance between rows / columns and for averaging. The corresponding barycenters of MMD and W_2 [CAD20, BGKL17] are used for averaging, and so the methods are coined kernel nearest neighbors (in short KernelNN) and Wasserstein nearest neighbors (in short W_2NN) respectively.

We elaborate on a vanilla version three step procedure of KernelNN, W_2NN that explicitly constructs neighborhoods. The input are measurements $\mathcal{Z}$, missingness $\mathcal{A}$, the target index (i,t) and the radius η,

Step 1: (Distance between rows) Calculate the distance between row i and any row $j \in [N] \setminus \{i\}$ by averaging the estimator of distribution metric $\widehat{\varrho}$:

$$\rho^{\mathsf{MMD}}_{i,j} := \frac{\sum_{s \neq t} A_{i,s}A_{j,s}\widehat{\mathsf{MMD}}^2_{\mathbf{k}}(Z_{i,s}, Z_{j,s})}{\sum_{s \neq t} A_{i,s}A_{j,s}} \quad \text{and} \quad \rho^{\mathsf{W}_2}_{i,j} := \frac{\sum_{s \neq t} A_{i,s}A_{j,s}\widehat{\mathsf{W}}^2_2(Z_{i,s}, Z_{j,s})}{\sum_{s \neq t} A_{i,s}A_{j,s}}.$$

Step 2: (Construct neighborhood) Construct a neighborhood of radius η within the tth column using the distances $\{\rho_{i,j} : j \neq i\}$:

$$\mathbf{N}^{\mathsf{MMD}}_{t,\eta} := \{j \in [N] \setminus \{i\} : \rho^{\mathsf{MMD}}_{i,j} \leq \eta\} \quad \text{and} \quad \mathbf{N}^{\mathsf{W}_2}_{t,\eta} := \{j \in [N] \setminus \{i\} : \rho^{\mathsf{W}_2}_{i,j} \leq \eta\}$$

Step 3: (Average across observed neighbors) Set $\mu^Z_{i,t} = n^{-1}\sum_{\ell=1}^n \boldsymbol{\delta}_{X_\ell(i,t)}$ as the empirical measure of the multiple measurements $Z_{i,t}$. Take the barycenter within the neighborhood:

$$\widehat{\mu}^{\mathsf{MMD}}_{i,t,\eta} := \frac{1}{|\mathbf{N}^{\mathsf{MMD}}_{t,\eta}|}\sum_{j \in \mathbf{N}^{\mathsf{MMD}}_{t,\eta}} A_{j,t}\mu^Z_{j,t} \quad \text{and} \widehat{\mu}^{\mathsf{W}_2}_{i,t,\eta} := \operatorname*{argmin}_{\mu} \sum_{j \in \in \mathbf{N}^{\mathsf{W}_2}_{t,\eta}} \mathsf{W}^2_2(\mu, \mu^Z_{j,t}).$$

For further details on the W_2 and MMD algorithms see [FCAD24] and [CFC$^+$24], respectively.

Algorithm 4: Vanilla (row-wise) distributional nearest neighbor

Input: $\mathcal{Z}, \mathcal{A}, \mathbf{k}, \eta, (i,t)$

Initialize entry-wise metric $\widehat{\varphi}(Z_{j,s}, Z_{j',s'}) \leftarrow \widehat{\mathsf{MMD}}^2_{\mathbf{k}}(Z_{j,s}, Z_{j',s'})$ or $\widehat{\mathsf{W}}^2_2(Z_{j,s}, Z_{j',s'})$

Initialize metric $\varphi(x,y) \leftarrow \mathsf{MMD}^2_{\mathbf{k}}(x,y)$ or $\mathsf{W}^2_2(x,y)$

Initialize tuning parameter $\eta \leftarrow (\eta_1, 0)$

Calculate row-wise metric $\left\{\rho^{\text{row}}_{i,j} : j \neq i\right\} \leftarrow \textsc{Distance}(\widehat{\varphi}, Z, A)$

Initialize weight $w_{j,s} \leftarrow \mathbf{1}(\rho^{\text{row}}_{i,j} \leq \eta_1, \rho^{\text{col}}_{s,t} \leq \eta_2)$

Calculate average $\widehat{\mu}_{i,t} \leftarrow \textsc{Average}(\varphi, \mathcal{W}, \mathcal{Z}, \mathcal{A})$

return $\widehat{\mu}_{i,t}$

C.6 Adaptively weighted nearest neighbors

We elaborate on the adaptive variant of the vanilla nearest neighbor algorithm AWNN as mentioned in App. C.2 and Tab. 1. The input are measurements $\mathcal{Z}$, and missingness $\mathcal{A}$. Note that there is no need for radius parameter η and hence no CV.

Step 1: (Distance between rows and initial noise variance estimate) Calculate an estimate for noise variance and then the distance between any pair of distinct rows $i, j \in [N]$ by averaging the squared Euclidean distance across overlapping columns:

$$\rho_{i,j} := \frac{\sum_{s \neq t} A_{i,s} A_{j,s} (Z_{i,s} - Z_{j,s})^2}{\sum_{s \neq t} A_{i,s} A_{j,s}}, \quad \overline{Z} \leftarrow \frac{\sum_{j,s \in [N] \times [T]} A_{j,s} Z_{j,s}}{\sum_{j,s \in [N] \times [T]} A_{j,s}}, \quad \text{and} \quad \widehat{\sigma}^2 \leftarrow \frac{\sum_{j,s \in [N] \times [T]} A_{j,s} (Z_{j,s} - \overline{Z})^2}{\sum_{j,s \in [N] \times [T]} A_{j,s}}$$

Step 2: (Construct weights) For all rows and columns $(i,t) \in [N] \times [T]$, evaluate $w^{(i,t)} = (w_{1,t}, \cdots, w_{n,t})$, the weights that optimally minimizes the following loss involving an estimate of the noise variance $\widehat{\sigma}^2$,

$$w^{(i,t)} = \arg\min_{\widehat{w}^{(i,t)}} \left[2 \log(2m/\delta) \widehat{\sigma}^2 \|\widehat{w}^{(i,t)}\|_2^2 + \sum_{i' \in [N]} \widehat{w}_{i',t} A_{i',t} \widehat{\rho}_{i',i} \right], \tag{3}$$

where $\widehat{w}^{(i,t)} = (\widehat{w}_{1,t}, \cdots, \widehat{w}_{n,t})$ is a non-negative vector that satisfy $\sum_{i'=1}^{n} \widehat{w}_{i',t} A_{i',t} = 1$.

Step 3: (Weighted average) Take the weighted average of measurements:

$$\widehat{\theta}_{i,t} = \sum_{i' \in [N]} \widehat{w}_{i',t} A_{i',t} X_{i',t}, \qquad \forall (i,t) \in [N] \times [T]$$

Step 4: (Fixed point iteration over noise variance) Obtain new estimate of noise variance and stop if difference between old and new $\widehat{\sigma}^2$ is small.

$$\widehat{\sigma}^2 \leftarrow \frac{1}{\sum_{i \in [N], t \in [T]} A_{i,t}} \sum_{i \in [N], t \in [T]} \left(Z_{i,t} - \widehat{\theta}_{i,t} \right)^2 A_{i,t}$$

No cross-validation in AWNN The optimization problem in (3) can be solved exactly in linear time (worst case complexity) using convex optimization [SPD25]. AWNN doesn't rely on radius parameter η. Not only it automatically assigns neighbors to $(i,t)^{th}$ entry during its weight calculation(non-neighbors get zero weight), but also takes into account the distance of the neighbors from the $(i,t)^{th}$ entry. The closer neighors get higher weights and vice - versa.

We further specify the exact implementation of the two modules DISTANCE, AVERAGE to recover AWNN:

Algorithm 5: AWNN for scalar nearest neighbor

Input: $\mathcal{Z}, \mathcal{A}, (i,t)$

Initialize entry-wise metric $\widehat{\varphi}(Z_{j,s}, Z_{j',s'}) \leftarrow (Z_{j,s} - Z_{j',s'})^2$ and metric $\varphi(x,y) \leftarrow (x-y)^2$

Initialize noise - variance estimate $\sigma_\epsilon^2 \leftarrow \text{Variance}\Big(\{Z_{i,t}\}_{(i,t)\in[N]\times[T]}\Big)$

Calculate row-wise metric $\big\{\rho_{i,j}^{\text{row}} : j \neq i\big\} \leftarrow \textsc{Distance}(\widehat{\varphi}, \mathcal{Z}, \mathcal{A})$

Initialize weight $\{w_{1,t}, \ldots, w_{n,t}\} \leftarrow \arg\min_{\widehat{w}^{(i,t)}} \Big[2\log(2m/\delta)\widehat{\sigma}^2 \|\widehat{w}^{(i,t)}\|_2^2 + \sum_{i'\in[N]} \widehat{w}_{i',t} A_{i',t} \widehat{\rho}_{i',i}\Big]$

Calculate average $\widehat{\theta}_{i,t} \leftarrow \textsc{Average}(\varphi, \mathcal{W}, Z, A)$

return $\widehat{\theta}_{i,t}$

C.7 New variant: Auto nearest neighbors

TSNN is a generalization of RowNN and ColNN by setting one of the tuning parameters to zero (see Tab. 1), whereas the idea underlying DRNN is fundamentally different from that of TSNN; DRNN debiases a naive combination of RowNN and ColNN whereas TSNN simply boosts the number of measurements averaged upon, thereby gaining from lower variance. So we simply interpolate the two methods for some hyper-parameter $\alpha \in [0,1]$; see Tab. 1. Notably the hyper-parameter η for both DRNN and TSNN are identical when interpolated.

Suppose $\mu_{i,t} = \theta_{i,t} + \varepsilon_{i,t}$ in (1) where $\varepsilon_{i,t}$ are centered i.i.d. sub-Gaussian distributions across i and t. When σ is large in magnitude, TSNN denoises the estimate by averaging over more samples, hence providing a superior performance compared to DRNN in a noisy scenario. When σ is small so that bias of nearest neighbor is more prominent, DRNN effectively debiases the estimate so as to provide a superior performance compared to TSNN. The linear interpolator AutoNN *automatically* adjusts to the underlying noise level and debiases or denoises accordingly; such property is critical when applying nearest neighbors to real world data set where the noise level is unknown. We refer to Fig. 3 for visual evidence.

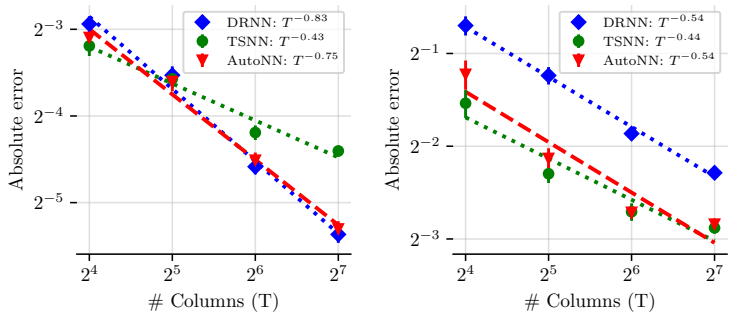


Figure 3: **Synthetic data experiments.** The data on the left has high signal-to-noise ratio, whereas the data on the right has low signal-to-noise ratio. See App. E.1 for details on the data generating process. Each point corresponds to the mean absolute error ± 1 standard error across 30 trials.

D Cross-Validation

For each nearest neighbor method, we use cross-validation to optimize hyperparameters including distance thresholds and weights, depending on which nearest neighbor algorithm is chosen. Specifically, for each experiment, we choose a subset of the training test to optimize hyperparameters by masking those matrix cells and then estimating the masked values. We utilize the HyperOpt library [BYC13] to optimize (possibly multiple) hyperparamters using the Tree of Parzen Estimator [BBBK11], a Bayesian optimization method. Our package supports both regular distance thresholds and percentile-based thresholds, which adapt to the distances calculated within the specific dataset.

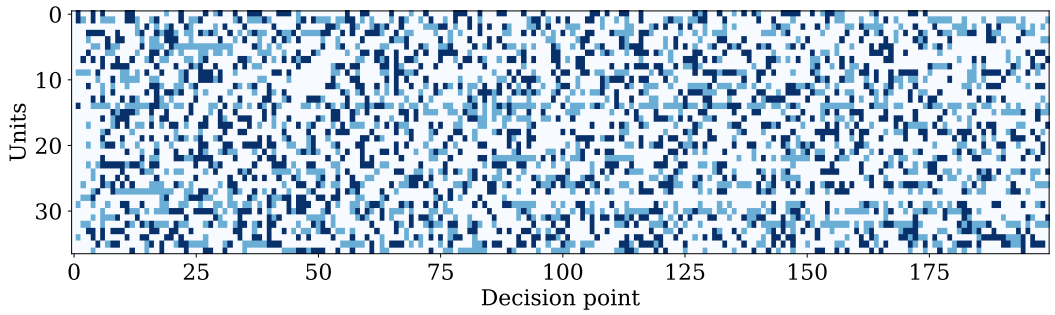


Figure 4: **HeartSteps V1 data notification pattern.** The dark blue entries indicate that the app sent a notification to a sedentary participant—the entry has value $A_{i,t} = 1$. The white entries indicate that the participant was available but did not receive a notification or they were active immediately prior to the decision point. The light blue entries indicate the participant was unavailable. We assign the value $A_{i,t} = 0$ for all the white and light blue entries.

E Case Study Details

The boxplots are generated using matplotlib's [Hun07] standard boxplot function. The box shows the first, second, and third quartiles. The bottom line shows the first quartile minus the 1.5× the interquartile range. The top line shows the third quartile plus 1.5× the interquartile range. All experiments are run on standard computing hardware (MacBook Pro with an M2 Pro CPU with 32 GB of RAM).

E.1 Synthetic data generation

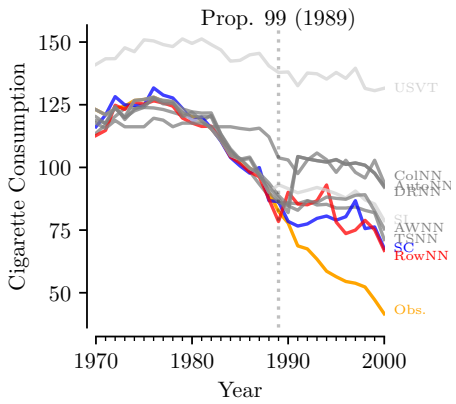
Generate $Z_{i,t} = X_{i,t} \sim N(\theta_{i,t}, \sigma^2)$, i.e., scalar matrix completion setting, with a linear factor structure $\theta_{i,t} = u_i v_t$. Row latent factors $u_i \in \mathbb{R}^4$ are i.i.d. generated across $i = 1, ..., N$, where each entry of u_i follow a uniform distribution with support $[-0.5, 0.5]$; column latent factors $v_t \in \mathbb{R}^4$ are generated in an identical manner. The missingness is MCAR with propensity $p_{i,t} = 0.5$ for all i and t. Further, the size of column and rows are identical $N = T$. For the left panel in Fig. 3, the noise level is set as $\sigma = 0.001$ and for the right panel $\sigma = 1$.

E.2 HeartSteps V1

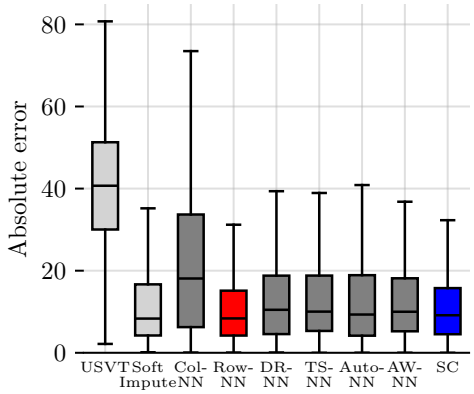
The mobile application was designed to send notifications to users at various times during the day to encourage anti-sedentary activity such as stretching or walking. Participants could be marked as unavailable during decision points if they were in transit or snoozed their notifications, so notifications were only sent randomly if a participant was available and were never sent if they were unavailable. To process the data in the framework of (1), we let matrix entry $Z_{i,t}$ be the average one hour step count for participant i and decision point t when a notification is sent (i.e. $A_{i,t} = 1$) and unknown when a notification is not sent (i.e. $A_{i,t} = 0$). The treatment assignment pattern is represented as the 37 x 200 matrix visualized in Fig. 4. We use the dataset downloaded from `https://github.com/klasnja/HeartStepsV1` (CC-BY-4.0 License).

E.3 MovieLens

We load MovieLens via a custom `MovieLensDataLoader` that (i) downloads and caches the `ml-1m.zip` archive, (ii) reads `ratings.dat` into a user × movie pivot table, and (iii) constructs the binary mask where observed entries correspond to rated user–movie pairs. The data matrix is $Z \in \{1, \ldots, 5\}^{6040 \times 3952}$ and mask matrix is $A \in \{0, 1\}^{6040 \times 3952}$. The data can be downloaded from `https://grouplens.org/datasets/movielens/1m/`. See `https://files.grouplens.org/datasets/movielens/ml-1m-README.txt` for the usage license.



(a) Synthetic controls for California in post-intervention period



(b) Absolute error on control states

Figure 5: **Nearest neighbor methods generate high-fidelity synthetic controls in counterfactual inference for panel data.**

E.4 Proposition 99

Next we consider a panel data setting, where our goal is to estimate the effect of the California Tobacco Tax and Health Protection Act of 1988 (a.k.a. Proposition 99) on annual state-level cigarette consumption[3]. By definition, the counterfactual cigarette consumption in California—had Proposition 99 never been enacted—is not observed. [ADH10] introduce the notion of a "synthetic control" to serve as a proxy for this unobserved value based on "neighboring" *control states* that never instituted a tobacco tax. These states are not close in a geographical sense, but rather close due to similarities in other covariates[4]. We take a different approach and use only the observed cigarette consumption levels from the control states, of which there are 38 in total. Thus, we frame our problem as a scalar matrix completion problem with $N = 39$ and $T = 31$ (see (1)). The last row in the matrix corresponds to the state of California.

Results & Discussion. For each method, we use a 64-16-20 train-validation-test split and use cross validation to fit any hyperparameters. Fig. 5 plots the various synthetic controls for California (left) and absolute error of each method on the 38 control states, for which we do observe the no-treatment values (right). From Fig. 5(a), we see that nearest neighbor methods, in particular TSNN and RowNN, are roughly on par with the gold-standard synthetic control method of [ADH10] ("SC") for estimating California's counterfactual cigarette consumption in the post-intervention period (after 1989). This is despite the fact that the nearest neighbor methods rely on less information for the estimation task. From Fig. 5(b), we see that all nearest neighbor methods, with the exception of ColNN, achieve

[3] measured as per capita cigarette sales in packs

[4] GDP per capita, beer consumption, percent aged 15–24, and cigarette retail prices

similar error levels as the synthetic control baseline. RowNN achieves even lower error levels. See supplementary experiment details in App. E.4.

Data comes primarily from the Tax Burden on Tobacco compiled by Orzechowski and Walker [OW23] (ODC-By License). Using synthetic control methods, Abadie et al. construct a weighted combination of control states that closely resembles California's pre-1988 characteristics and cigarette consumption patterns. The optimal weights produce a synthetic California primarily composed of Colorado (0.164), Connecticut (0.069), Montana (0.199), Nevada (0.234), and Utah (0.334), with all other states receiving zero weight. The treatment effect is estimated as the difference between actual California per-capita cigarette sales and those of synthetic California after Proposition 99's implementation. By 2000, this analysis revealed that annual per-capita cigarette sales in California were approximately 26 packs lower than what they would have been without Proposition 99, representing about a 25% reduction in cigarette consumption. To validate these findings, the authors conducted placebo tests by applying the same methodology to states not implementing tobacco control programs, confirming that California's reduction was unusually large and statistically significant (p = 0.026).

Proposition 99, the California Tobacco Tax and Health Protection Act of 1988, dataset spans from 1970 to 2000, providing 19 years of pre-intervention data before Proposition 99 was implemented in 1988 and 12 years of post-intervention data. It provides annual state-level cigarette consumption measured as per capita cigarette sales in packs based on tax revenue data. This data serves as a real data benchmark for many of the variants of synthetic controls [ABD$^+$21]. We use the CDC dataset for the Nearest Neighbors methods and only use the target variable (i.e., cigarette consumption measured in packs per capita), and the dataset from SyntheticControlMethods library[5] for the SC baseline, since it relies on additional covariates.

E.5 PromptEval

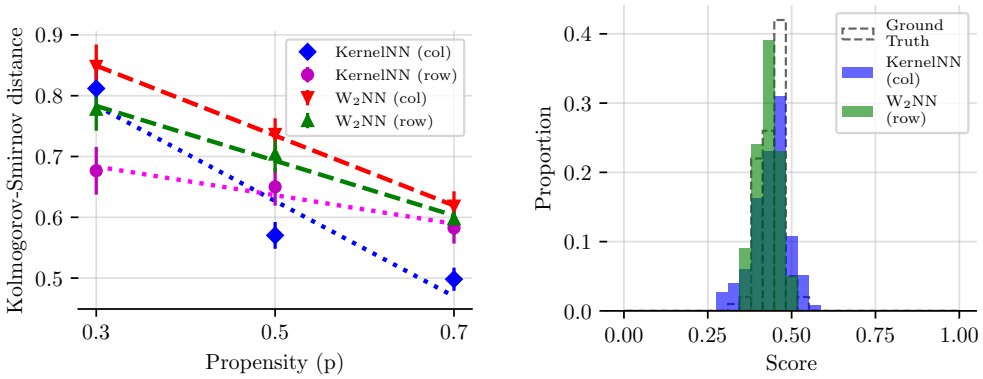
E.6 Efficient LLM evaluation: PromptEval

The rapid advancement of LLMs have placed them at the center of many modern machine learning systems, from chatbots to aids in medical education [GHC$^+$25]. In practice, system architects want to strike the right balance of real-world performance and cost, but navigating this Pareto frontier is a daunting task. 2024 alone saw at least 10 new models from Anthropic, Google, Meta, and OpenAI, not even counting the multitude of open-source fine-tuned models built on top of these. On specific tasks, smaller, fine-tuned models may even outperform the latest frontier models, in addition to being more cost effective.

We investigate how matrix completion, specifically nearest neighbor methods, can alleviate some of these burdens. We use the PromptEval dataset [PXW$^+$24], which evaluates 15 open-source language models (ranging in size from 3B to 70B parameters) and 100 different prompting techniques across the 57 tasks of the MMLU benchmark [HBB$^+$20]. In practice, the performance of a model depends—sometimes dramatically—on the precise input prompt. This suggests that we need to consider the performance of a model across a wide range of prompts, rather than any one prompt in particular. Thus, we model this problem as a distributional matrix completion problem with $N = 15$, $T = 57$, and $n = 100$. Given one of 57 tasks, we aim to accurately characterize the performance of each model without resorting to exhaustive evaluation. Nearest neighbors leverage commonalities across models and tasks to estimate the performance distribution of each entry, which was otherwise not considered in [PXW$^+$24]; previous literature achieves efficient evaluation per model and task in isolation without leveraging any across model / task information.

Results & Discussion. We randomly include each entry in the matrix independently with probability $p \in \{0.3, 0.5, 0.7\}$ and impute the missing entries using the KernelNN and W_2NN methods of Tab. 1. For each method, we consider both the the row-wise and column-wise variants. Fig. 6(a) reports the mean Kolmogorov-Smirnov (KS) distance between the imputed and ground-truth distributions across the entries in the test set for varying missingness values. As expected, estimation error decreases as p increases. Fig. 6(b) visualizes the imputed distributions using row-wise KernelNN and column-wise W_2NN (at $p = 0.7$) for a select entry, along with the ground-truth distribution. Even with 30% of matrix entries missing, distributional NN methods are able to recover the underlying distribution.

[5] `https://github.com/OscarEngelbrektson/SyntheticControlMethods/tree/master` (Apache-2.0 License)



(a) Mean KS distance between estimated and ground-truth distributions

(b) KernelNN vs. W_2NN

Figure 6: **Distributional nearest neighbor methods enable efficient LLM evaluation on MMLU.** We estimate LLM score distributions across all models and tasks given only a limited number of model-task evaluations, determined by the propensity p. See App. E.6 for a detailed discussion.