# Web Supplement: Deep learning for the dynamic prediction of multivariate longitudinal and survival data

## 1 DeepSurv Supplemental Details

The outline of the DeepSurv model is presented in Figure S1. The figure shows that the DeepSurv architecture closely follows that of a standard feed-forward neural network. Each linear layer is followed by an activation function. In our implementation, the rectified linear unit (ReLU) was used, where $ReLU(x) = \max(0, x)$. This is followed by dropout, a regularization technique where during training some nodes are randomly omitted to avoid overfitting. Lastly, batch normalization re-centers and re-scales the layer output to allow for faster and more stable training. When used in conjunction with MFPCA, the MFPC scores derived from longitudinal outcomes are passed into the network together with the baseline covariates. Hyperparameters (i.e., the number of hidden layers, the number of nodes per hidden layer) were selected using a simple grid search. The hyperparameters and their selection ranges are listed in Table S1. In determining the selection ranges, it is helpful to refer to the ranges used in the DeepSurv paper. Typically, one can start with one or two hidden layers and increase the number if necessary. The number of nodes per hidden layer is often close to the number of

| Hyperparameter | Selection Range |
|---|---|
| # Hidden Layers | 1, **2**, 3, 4 |
| # Nodes per Hidden Layer | 16, 32, **64** |
| Dropout Probability | **0.2**, 0.4 |
| Activation Function | ReLU |
| # Epoch | 10, **12**, 14, 16 |

Table S1: Hyperparameters selection ranges/values for DeepSurv. Values selected for the simulation study are marked in bold.

input features. An alternative approach is to first create a network that is large enough to overfit but can perform the task reasonably well. This will shed light on the upper bound of the selection ranges.

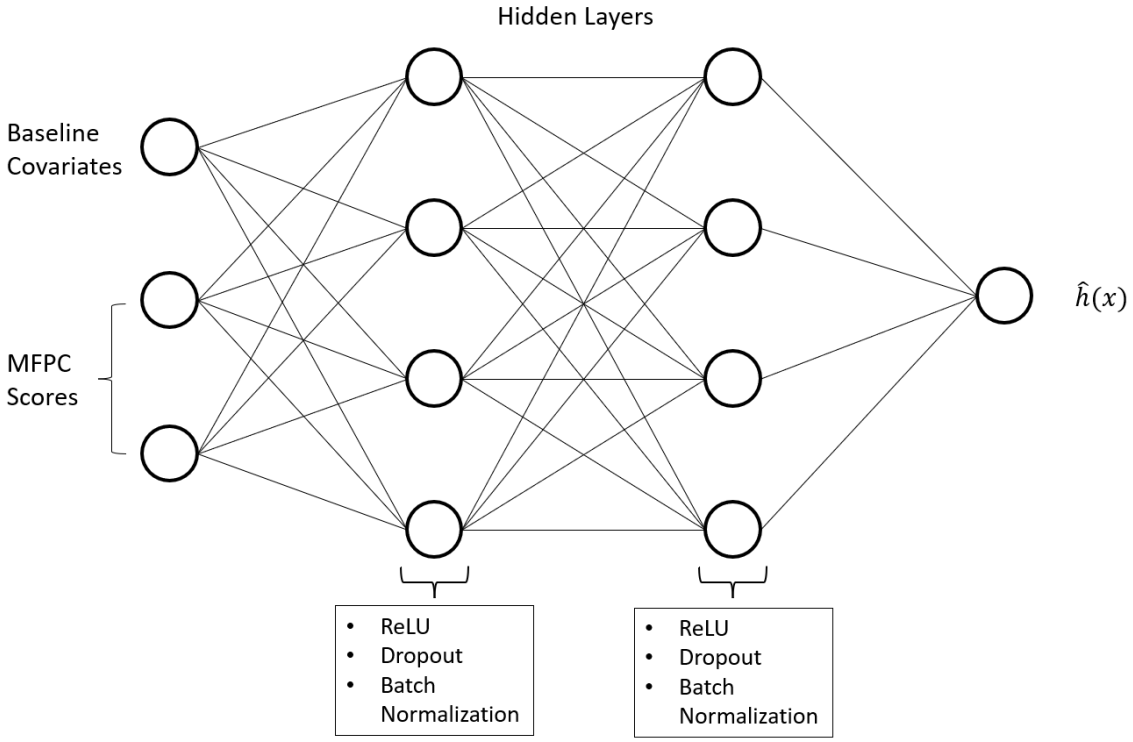



Figure S1: DeepSurv neural network with two hidden layers. In MFPCA-DS, MFPC scores are passed together with baseline covariates as network inputs.

## 2 MATCH-net Supplemental Details

The major components of the MATCH-net architecture are described in the main text and illustrated in Figure S3. The longitudinal data is processed using convolutional layers. A missingness mask, the same size as the longitudinal data, indicates the positions where the longitudinal data was observed. The mask is processed with its own convolutional layers in parallel and concatenated to the output of the main convolutional layer. In addition, ReLU, dropout, and batch normalization are used after each convolutional and fully connected layers. Prior to the fully connected layers, a global average pool operation is used, which takes the average of each feature map. The average of each feature map becomes an input node to the feed-forward neural network. The hyperparameters of MATCH-net were selected from a grid search, with the selection ranges listed in Table S2. In determining the selection ranges, it is helpful to refer to the ranges used in supplement of the MATCH-net paper.

| Hyperparameter | Selection Range |
|---|---|
| # Convolutional Layers | 2, 3, **4**, 5 |
| # Filters per Conv Layer | 16, **32**, 64 |
| # Filters per Mask Conv Layer | **8**, 16, 32 |
| # FF Hidden Layers | 1, **2**, 3, 4 |
| # Nodes per FF Hidden Layer | 16, 32, **64** |
| Convolutional Dropout Probability | 0.2, **0.4** |
| FF Dropout Probability | **0.2**, 0.4 |
| Activation Function | ReLU |
| # Epoch | 10, 15, 20, **25**, 30 |

Table S2: Hyperparameters selection ranges/values for MATCH-net. Values selected for the simulation study are marked in bold.

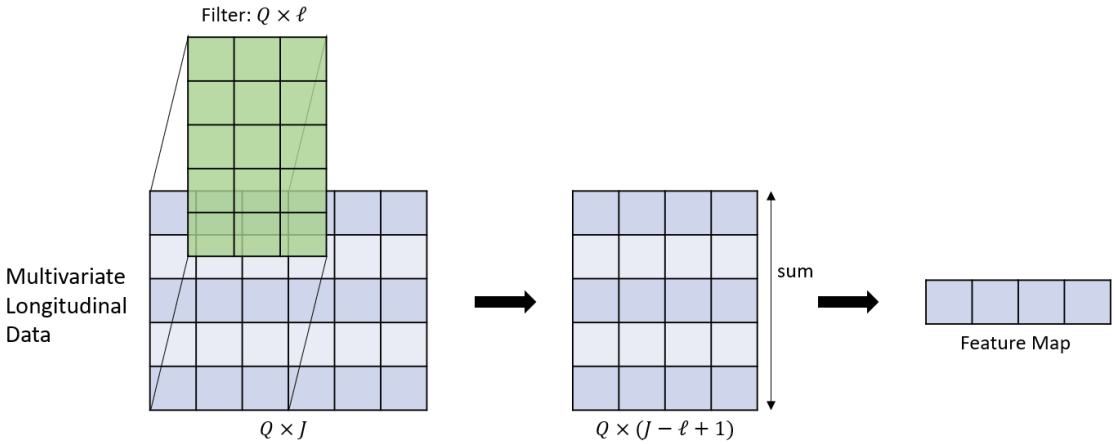



Figure S2: Illustration of a single convolution operation, where $Q$ is the number of longitudinal outcomes, $J$ the number of visits, and $\ell$ the length of the filter.

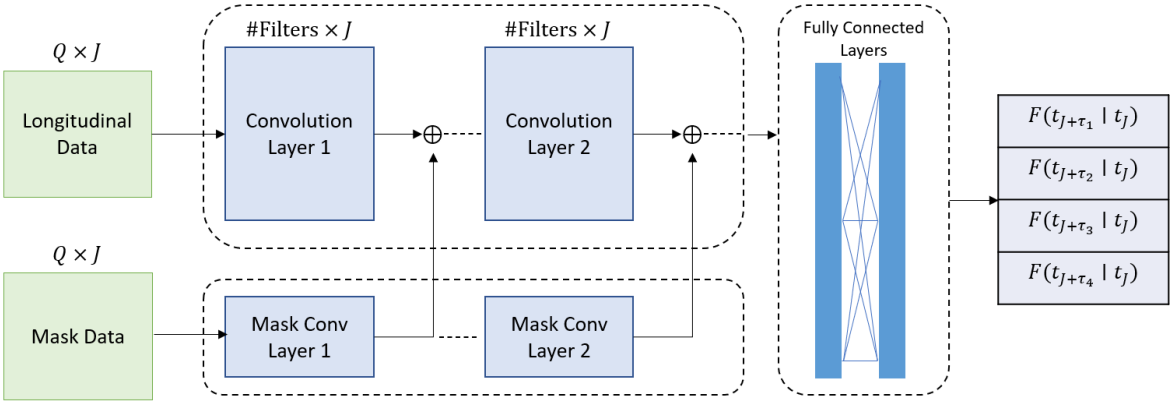



Figure S3: MATCH-net architecture which processes the longitudinal outcomes and the missingness information using parallel convolutional layers. After each convolutional layer, the output of the mask layer is concatenated to the output of the main layer for longitudinal data.

# 3 TransformerJM Supplemental Details

## 3.1 Attention and Positional Encoding

Attention is a function which takes in a query and a set of key-value pairs. The query, $\boldsymbol{Q}$, is a transformation of the first sequence by multiplying by a weight matrix, $\boldsymbol{W}^Q$. Likewise, the keys, $\boldsymbol{K}$, and values, $\boldsymbol{V}$, are transformations of the second sequence by using the respective weight matrices $\boldsymbol{W}^K$ and $\boldsymbol{W}^V$. These weight matrices are the parameters to be estimated by the network. Attention is computed as the weighted sum of the values, where the weights are determined by the similarity between the queries and keys. This is often given as $Attention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = softmax\left(\frac{\boldsymbol{Q}^T\boldsymbol{K}}{\sqrt{d_k}}\right) \cdot \boldsymbol{V}$, where similarity is measured as the dot product between the queries/keys and scaled by $d_k$, the dimensions of the key vector. Self-attention refers to when the same sequence is used to derive both the query and the key-value pairs. This allows the network to understand how different parts of a sequence are interrelated.

To prevent positions from attending to missing time points or to future time points which have not yet occurred, illegal connections can be masked out when computing the attention. This is done by adding a mask $\boldsymbol{M} = (-\infty \text{ if masked}, 0 \text{ otherwise})$ to the query/key dot product.

$$MaskedAttention = softmax\left(\frac{\boldsymbol{Q}^T\boldsymbol{K} + \boldsymbol{M}}{\sqrt{d_k}}\right) \cdot \boldsymbol{V}$$

It is often beneficial to calculate attention multiple times to give several different representations of the data. Referred to as multi-head attention, attention is calculated multiple times in parallel. The resulting attention heads are concatenated and multiplied by an additional weight matrix to reduce the dimensions to the original size.

The attention mechanism learns dependencies between all allowable time points; how-

ever, it does not account for the ordering of the sequence. Information about the position or time stamp of the sequence can be instilled through positional encodings. For a sequence of vectors of length $d\_model$, the positional information for each vector can be added through element-wise addition of a vector representing a given position. While there are different options for positional encodings, using sine and cosine functions of different frequencies is a common choice for transformers. The positional encoding for time $t$ is a vector of size $d\_model$, where even and odd indices are calculated using sine and cosine respectively.

$$PE = \begin{bmatrix} \sin(t/10000^{i/d\_model}) & \text{, where i is even} \\ \cos(t/10000^{i/d\_model}) & \text{, where i is odd} \end{bmatrix}$$

Intuitively, the positional encoding for two time points close together should be very similar for all indices of the vector. On the other hand, the positional encoding would be distinct for two time points far apart. Whereas MFPCA and CNN's require the observation times to be on a fixed grid, positional encoding differs by directly embedding the time into the data.

## 3.2 Training Details

As a auto-regressive model, the transformer generates predictions sequentially, feeding the model output back in as input for the next time step. However, during training, a technique known as teacher forcing is used to speed up the training time by allowing for more parallelization. In teacher forcing, the model output for a given time step is replaced with the true observed values as the input for generating the next time step. Teacher forcing provides two main benefits. First, predictions for different times can be made in parallel since they are calculated based on the true observed sequence and not based on past predictions. Second, teacher forcing allows for faster convergence during

training since errors are not sequentially propagated, but instead corrected at each time step.

The Transformer has several distinct hyperparameters. First, the number of decoder blocks determines the number of times attention is calculated. Transformers that are "deep" have a higher number of blocks to model more complex sequential data. The number of attention heads also increases with the complexity of the data. Lastly, *d_model* is the length of the vector representing an element of the sequence. Its length should be reflective of the complexity of individual elements in comparison to the number of decoder blocks or the number of attention heads which accounts for the complexity between elements. The list of hyperparameters for TransformerJM is listed in Table S3. In determining the selection range, we can used standard transformers for natural language processing as a reference. For example, GPT-2, a common language transformer, starts out with 12 decoder blocks, 12 attention heads, and *d_model* = 768. Because modeling natural language is a more complex task than modeling longitudinal clinical assessments, we can limit our selection range using the GPT-2 hyperparameters as a recommended upper bound. In addition, the parameter *d_model* should be reflective of the number of longitudinal outcomes/baseline variables. Hence, we limited the selection range to be in a similar magnitude. In contrast, *d_model* for language models should be much larger to represent the many possible meanings of the words in a language.

| Hyperparameter | Selection Range |
|---|---|
| # Decoder Blocks | 3, 5, **7**, 9 |
| d_model | 16, **32**, 64 |
| # Attention Heads | 2, **4**, 8 |
| Dropout Probability | **0.1**, 0.2 |
| Activation Function | ReLU |
| # Epoch | 10, 15, 20, **25**, 30 |

Table S3: Hyperparameters selection ranges/values for TransformerJM. Values selected for the simulation study are marked in bold.

## 3.3 Scalability and Computational Complexity

Multivariate longitudinal data is comprised of three main dimensions: the number of subjects, the number of visits, and the number of longitudinal outcomes. We briefly discuss how the Transformer scales in terms of these three dimensions. Table S4 reports the computational times of several simulations, each modifying one of the aforementioned dimensions. The baseline setting uses 1,000 subjects, 10 visits, and $d\_model = 32$ (where $d\_model$ is the length of the vector containing the information at a given visit). First, increasing the number of subjects will increase the computational cost linearly. Since neural networks are not trained using all the data simultaneously but rather processed in batches, having additional subjects increases the number of batches which need to be processed. Next, increasing the number of visits will increase the computational cost quadratically. This is due to the self-attention mechanism which computes the relation of every time point to all other time points. This is generally not a concern for longitudinal measures of clinical assessments where there are a modest number of time points. However, some modifications might be required for cases with intensive longitudinal data i.e., where measurements are collected frequently and automatically by digital devices such as accelerometers or biosensors. Lastly, having more longitudinal outcomes/baseline variables may require a higher $d\_model$ to contain the increased amount of information. Increasing $d\_model$ will increase the computational cost linearly. The computational complexity of the self-attention layer is given by $\mathcal{O}\left(J^2 \cdot d\_model\right)$ where $J$ is the number of visits.

| Setting | Computational Time (Seconds) | Growth Rate |
|---|---|---|
| Baseline | 131.2 | — |
| # Subjects = 20,000 | 1808.1 | Linear |
| # Subjects = 50,000 | 4633.0 | Linear |
| # Visits = 20 | 145.6 | Quadratic |
| # Visits = 50 | 328.7 | Quadratic |
| d_model = 64 | 139.7 | Linear |
| d_model = 128 | 147.2 | Linear |

Table S4: Scalability of TransformerJM. Baseline has a setting of #Subjects = 1,000, #Visits = 10, and d_model = 32. Each subsequent setting makes only the indicated changes to the baseline setting.

# 4 Simulation

## 4.1 Simulation Procedure

We created 100 simulated datasets, each with a sample size of $I = 1000$ subjects. Each subject had $Q = 3$ longitudinal covariates and a maximum of $J_i = 11$ annual visits from observation times $t_{ij} = [0, 10]$. The true longitudinal trajectory, $X_{iq}(t)$, was simulated from the submodel:

$$X_{iq}(t) = \beta_{0q} + \beta_{1q} x_{iq} + \beta_{tq} t + b_{iq} \ .$$

The observed longitudinal data is given by $Y_{iq}(t_{ij}) = X_{iq}(t_{ij}) + \epsilon_{iq}(t_{ij})$ for $j = 1 \dots J$, where $\epsilon_{iq}(t_{ij}) \sim N(0, 1)$. The coefficients were set as $\beta_{0q} = [1.5, 2, 0.5]$, $\beta_{1q} = [2, -1, 1]$, and $\beta_{tq} = [1.5, -1, 0.6]$. The scalar covariate was generated from a random normal distribution, $x_{iq} \sim N(3, 1)$, and the subject-specific random effects was generated from a multivariate normal distribution, $b_{iq} \sim MVN(0, \Sigma)$, where

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \eta_{12}\sigma_1\sigma_2 & \eta_{13}\sigma_1\sigma_3 \\ & \sigma_2^2 & \eta_{23}\sigma_2\sigma_3 \\ & & \sigma_3^2 \end{bmatrix} ,$$

with $[\sigma_1, \sigma_2, \sigma_3] = [1, 1.5, 2]$ and $[\eta_{12}, \eta_{13}, \eta_{23}] = [-0.2, 0.1, -0.3]$.

The survival submodel was defined by the hazard function

$$h_i(t) = h_0(t) \exp\left[\boldsymbol{\gamma}\boldsymbol{Z}_i + \sum_{q=1}^{Q} \boldsymbol{\alpha}_q X_{iq}(t)\right] . \tag{1}$$

We chose a constant baseline hazard function, $h_0(t) = \exp(-7)$. Two time-independent covariates were generated and denoted as $\mathbf{Z}_i = [z_{i1}, z_{i2}]$, where $z_{i1} \sim \text{Bin}(p = 0.5)$ and $z_{i2} \sim N(0, 1)$. The coefficients for the time-independent covariates and the true longitudinal trajectories were set as $\boldsymbol{\gamma} = [-4, -2]$ and $\boldsymbol{\alpha} = [0.2, -0.2, 0.4]$ respectively. Survival times were generated by passing the standard uniform distribution through the inverse probability integral transformation of the survival function, $S_i(t) = \exp[-\int_0^t h_i(u)du]$. The censoring times were simulated independently of survival times from a uniform distribution, $U(3, 25)$, giving a censoring rate of approximately 30%. We refer to this first simulation setting as Scenario 1. To demonstrate the effects of model misspecification, an interaction term was added to the baseline covariates, $\boldsymbol{Z}_i = [z_{i1}, z_{i2}, z_{i1} \cdot z_{i2}]$ in Equation (1) with corresponding coefficients $\boldsymbol{\gamma} = [-4, -2, 3]$. When training the models, the interaction term was not specified as a predictor. This simulation setting is referred to as Scenario 2 and evaluates the models' ability to capture the effect of the interaction term without needing explicit input. Lastly, a time-dependent covariate was introduced to create a simulation setting with non-proportional hazards. This was done by adding a time-varying coefficient to $z_{i2}$, resulting in $\boldsymbol{\gamma} = [-4, -2 + 3\sin(t)]$. This last simulation setting is referred to as Scenario 3.

## 4.2 Code

The code for the model implementations and the simulation study can be found at: `https://github.com/reylined/TransformerJM`.

## 4.3 Additional Results: Simulation

| | True iAUC | MFPCA-Cox | MFPCA-DS | MATCH-net | TransformerJM |
|---|---|---|---|---|---|
| Scenario 1 | 0.926 | 0.923 | 0.920 | 0.917 | 0.920 |
| Scenario 2 | 0.922 | 0.855 | 0.914 | 0.912 | 0.915 |
| Scenario 3 | 0.950 | 0.877 | 0.905 | 0.928 | 0.941 |

(a) Integrated AUC

| | True iBS | MFPCA-Cox | MFPCA-DS | MATCH-net | TransformerJM |
|---|---|---|---|---|---|
| Scenario 1 | 0.091 | 0.095 | 0.097 | 0.107 | 0.097 |
| Scenario 2 | 0.084 | 0.120 | 0.091 | 0.098 | 0.091 |
| Scenario 3 | 0.087 | 0.151 | 0.128 | 0.127 | 0.097 |

(b) Integrated Brier Score

Table S5: Predictions were made at $\Delta t = 2$ into the future. Integrated AUC and Brier score calculated for three scenarios: baseline (Scenario 1), unspecified interactions (Scenario 2), and non proportional hazards (Scenario 3). The RMSE of the longitudinal predictions is not included, as the change to $\Delta t = 2$ is only applicable to the survival predictions.

# 5 Additional Results: Alzheimer's Disease Datasets

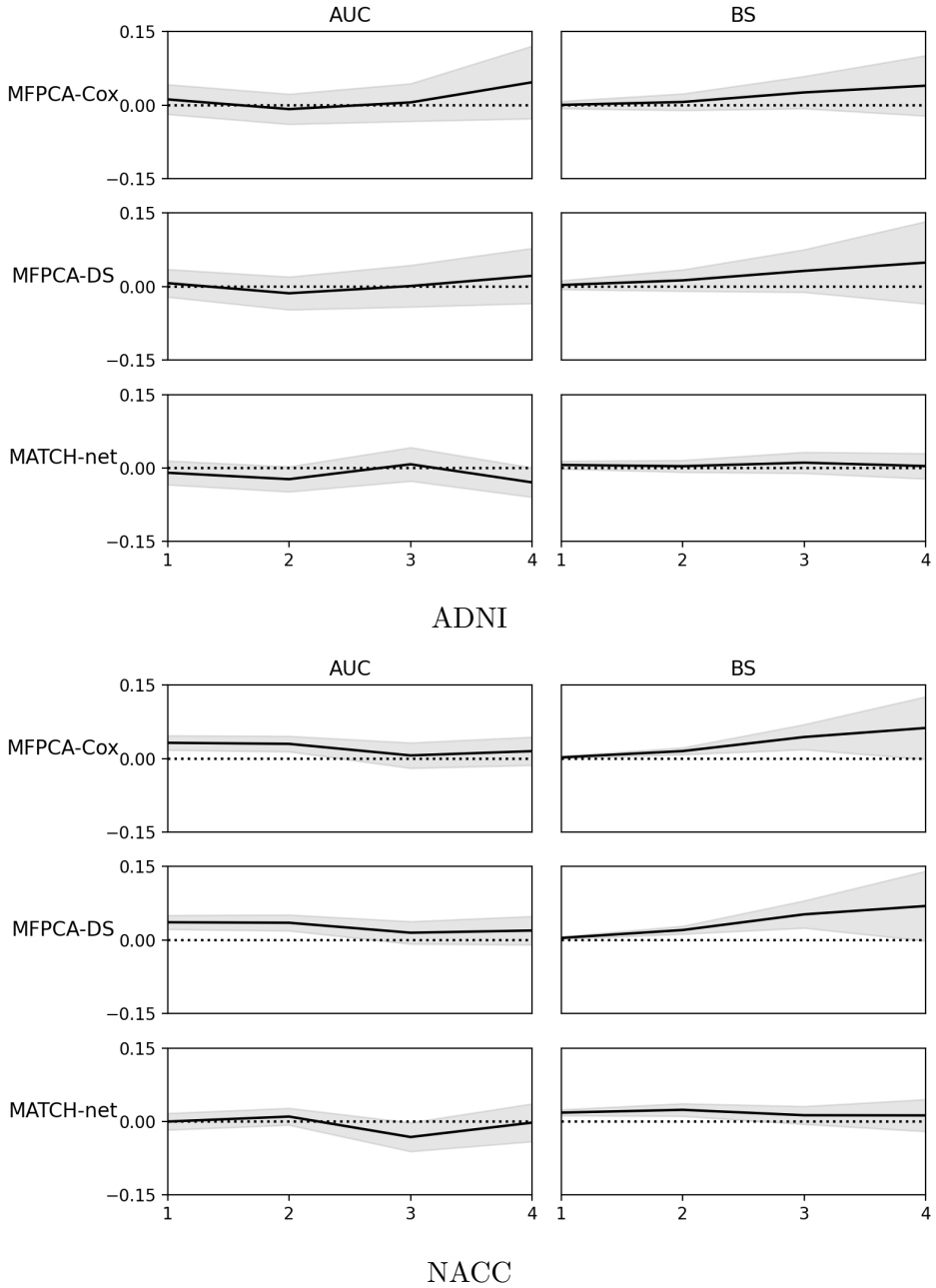
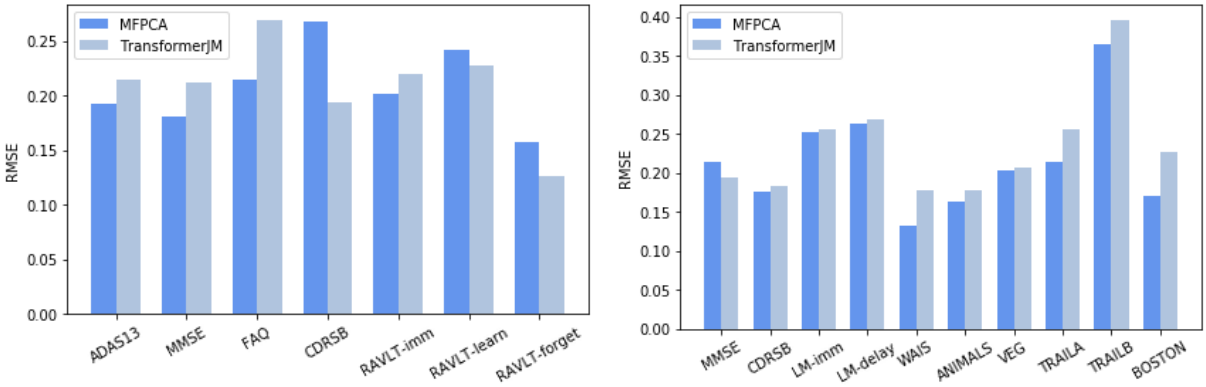



Figure S4: The 95% pointwise confidence intervals for the difference in TransformerJM's AUC and Brier score in comparison to other methods at landmark times [1,2,3,4] and $\Delta t = 1$. A positive difference indicates better performance for TransformerJM for both AUC and Brier score. Difference in AUC is calculated as $AUC_{\text{Transformer}} - AUC_{\text{other}}$ and difference in Brier score is calculated as $BS_{\text{other}} - BS_{\text{Transformer}}$.




ADNI NACC

Figure S5: RMSE of longitudinal outcomes for the ADNI and NACC datasets.