Graph Structure Inference with BAM: Introducing the Bilinear Attention Mechanism - APPENDIX

A ARCHITECTURAL DETAILS

A.1 SPD Activation Function

To obtain an activation function for the symmetric positive semi-definite (SPD) net, we leverage the following theorem, a direct consequence of (Theorem 4.11 Guillot and Rajaratnam, 2015), which is based on the work of Schoenberg (1942):

Theorem 1. *Any continuous function from $C([-1,1],[-1,1])$ acting elementwise on a matrix preserves positive definiteness if it can be expressed as a series $f:[-1,1]\to[-1,1]$ with*

$$f(x)=\sum_{k=1}^{\infty} x^k w_k,$$

$$\text{subject to}\quad \sum_{k=1}^{\infty} w_k \leq 1 \quad \text{with} \quad w_k \geq 0, \quad \forall k \in \mathbb{N}.$$

We employ this theorem to construct an activation function for the SPD neural network, using a relatively small maximal polynomial degree value of $N_{\max}=3$. This function employs trainable weights w_k and is applied after 'correlation normalization', i.e., the conversion of covariance matrices into correlation matrices. Trainable activation functions using low-degree Taylor polynomials were also proposed in (Chung et al., 2016) for general neural networks, not focusing on SPD data. Additionally, (Apicella et al., 2021) provides various types of trainable activation functions for Euclidean neural networks.

A.2 Observational Attention

This layer accepts an input $\boldsymbol{X} \in \mathbb{R}^{M\times d\times C}$, from which it generates keys $\boldsymbol{K}=\boldsymbol{X}\boldsymbol{W}_K \in \mathbb{R}^{M\times d\times c}$, queries $\boldsymbol{Q}=\boldsymbol{X}\boldsymbol{W}_Q \in \mathbb{R}^{M\times d\times c}$, and values $\boldsymbol{V}=\boldsymbol{X}\boldsymbol{W}_V \in \mathbb{R}^{M\times d\times C}$, with $\boldsymbol{W}_K, \boldsymbol{W}_Q \in \mathbb{R}^{C\times c}$, and $\boldsymbol{W}_V \in \mathbb{R}^{C\times C}$.

For each $m=1,\ldots,M$, keys and queries are combined in parallel along the inner axis, leading to

$$\boldsymbol{K}\odot\boldsymbol{Q} := \left(\boldsymbol{K}_{m,\cdot,\cdot}\boldsymbol{Q}_{m,\cdot,\cdot}\right)_{m=1,\ldots,M} \in \mathbb{R}^{M\times d\times d}.$$

This results in the attention weights

$$\boldsymbol{A}=\sigma\left(\frac{\boldsymbol{K}\odot\boldsymbol{Q}^{T(1,3,2)}}{\sqrt{c}}\right) \in \mathbb{R}^{M\times d\times d},$$

where $T(\text{perm})$ denotes a permutation of the axes according to a permutation perm and σ is the softmax operation along the last axis. For example, if perm $=[1,3,2]$, the operation would rearrange the second and third axes of the tensor while the first axis stays unchanged. Finally, for each m, the output is computed as

$$\boldsymbol{H}=\boldsymbol{A}\odot\boldsymbol{V} \in \mathbb{R}^{M\times d\times C}.$$

A.3 Residual Connections and Normalization.

Normalization is critical in attention networks (Xiong et al., 2020), with various methodologies available (Wu and He, 2018; Ba et al., 2016). We adopt normalization and a residual connection (He et al., 2016) on the input of all attention layers, expressed as $\boldsymbol{S} + \text{Attention}(\text{LayerNorm}(\boldsymbol{S}))$. Particularly when $M < d$, the residual connection can enhance the rank of covariance matrices, so full-rank representations can be attained even for under-determined problems. For learnable residual scaling, we utilize methods from Touvron et al. (2021); Bachlechner et al. (2021).

Within the SPD manifold, we employ correlation normalization alongside residual connections. This approach preserves positive definiteness and corresponds intuitively to standard normalization in Euclidean space.

A.4 Multiple Heads

In all attention layers, we employ multihead attention, a process that divides the input tensor along the channel axis into several smaller tensors. Each of these is then subjected to attention independently.

B THEORETICAL FOUNDATION FOR THE THREE-CLASS EDGE CLASSIFICATION PROBLEM

Building upon the foundational principles of Markov and faithfulness (Koller and Friedman, 2009), we demonstrate that the three-class classification problem can be theoretically deduced from the distribution of the nodes by examining the following independence relations: If there is an directed edge from node X to Y in the directed acyclic graph (DAG), then X and Y are dependent given any set from the power set of the other nodes, i.e.:

$$X \to Y \implies X \not\perp\!\!\!\perp Y | \boldsymbol{C} \quad \forall \boldsymbol{C} \in \mathcal{P}(\mathcal{V} \setminus \{X, Y\}).$$

For an immorality between X and Y, there exists a set of nodes $\boldsymbol{C} \in \mathcal{P}(\mathcal{V} \setminus \{X, Y\})$ within the power set of all other nodes such that X and Y are conditionally independent given this set (e.g., the set of all common ancestors of X and Y, or the set of all parents of X or Y), but dependent given all other nodes in the graph, i.e.:

$$X \to Z \leftarrow Y, X \not\leftarrow Y, X \not\to Y \implies \exists \boldsymbol{C} \in \mathcal{P}(\mathcal{V} \setminus \{X, Y\}) : X \perp\!\!\!\perp Y | \boldsymbol{C}, X \not\perp\!\!\!\perp Y | \mathcal{V} \setminus \{X, Y\}.$$

If there is no edge between X and Y, and if X and Y do not have a common child, then X and Y are conditionally independent given all other nodes, i.e.,

$$X \not\leftarrow Y, X \not\to Y, \quad \nexists Z : X \to Z \leftarrow Y \quad \implies X \perp\!\!\!\perp Y | \mathcal{V} \setminus \{X, Y\}.$$

Note that testing for the no-edge class is cost-effective, as one only needs to test for a single set, rather than checking for any $Z \in \mathcal{V} \setminus \{X, Y\}$ if there is a v-structure $X \to Z \leftarrow Y$. Testing for a sepset to differentiate between the skeleton and moralized edge classes is more intricate. The neural network is tasked with learning an approximation for this distinction.

C DETAILS OF THE CPDAG ESTIMATION MODEL

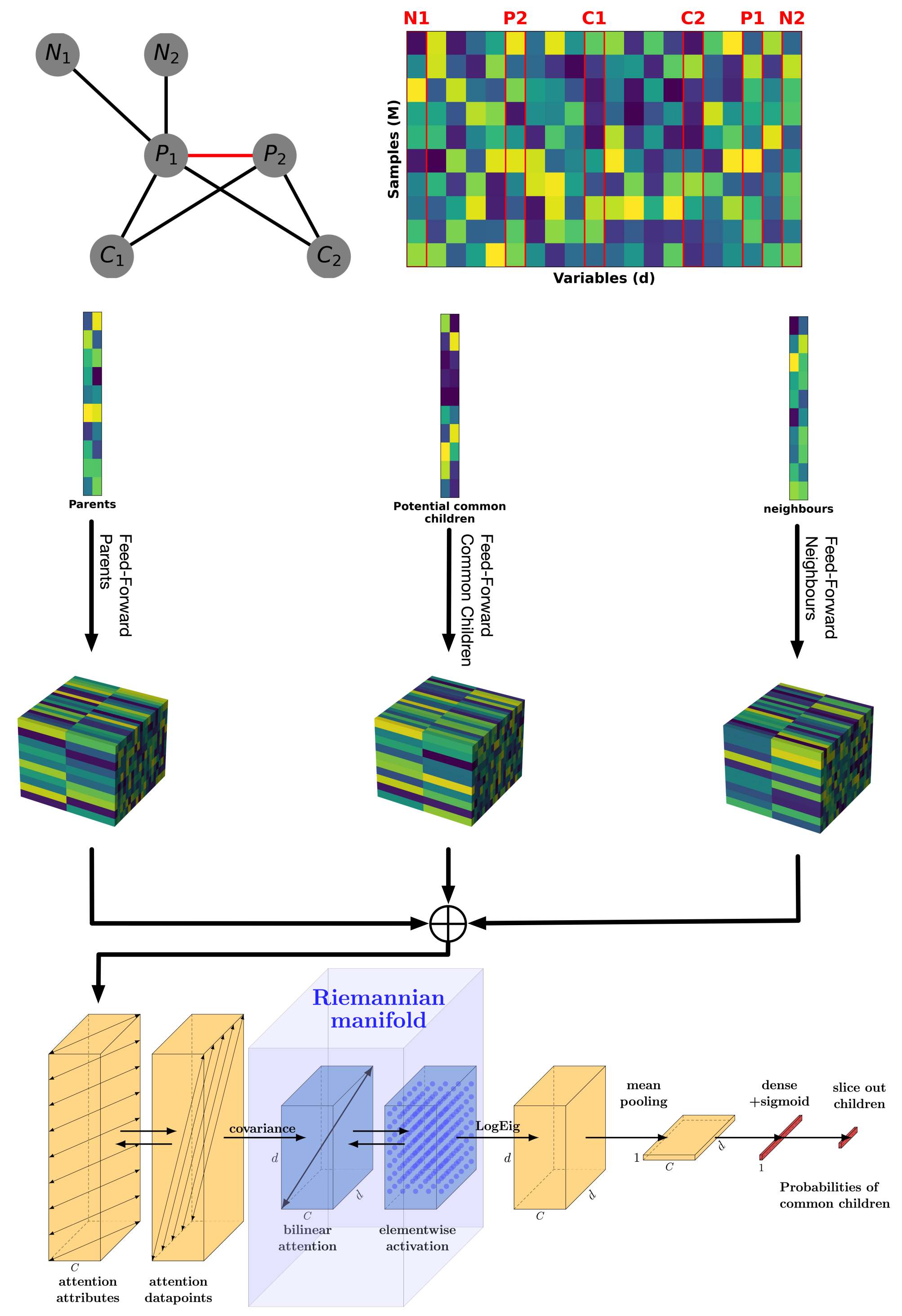


Figure 1: CPDAG Estimation Architecture: In the graph, black edges represent undirected connections, whereas the red edge signifies an immorality. Columns in the data matrix corresponding to parent nodes, potential common children, and neighboring nodes are taken as input for the neural network and are processed through three distinct feed-forward networks. The embeddings are concatenated along the variable axis and processed through a network with data matrix attention, bilinear attention, and a LogEig layer. Mean pooling and sigmoid activation are applied to output probabilities for potential common children.

To estimate a CPDAG from the graph skeleton, along with the set of immoralities between pairs of nodes, we test each estimated immorality to determine which potential common child nodes are indeed common children. The parent nodes, denoted by pa, are the nodes between which an immorality was first estimated. Potential common child nodes, denoted by cc, are nodes that have an edge to both parent nodes. Neighbor nodes, denoted by ne, are nodes that have one edge to exactly one parent.

We take the columns $\boldsymbol{x}_{\mathrm{pa}} \in \mathbb{R}^{M\times 2}, \boldsymbol{x}_{\mathrm{cc}} \in \mathbb{R}^{M\times|\mathrm{cc}|}, \boldsymbol{x}_{\mathrm{ne}} \in \mathbb{R}^{M\times|\mathrm{ne}|}$ of the data matrix $\boldsymbol{X}$ corresponding to $\mathrm{pa}, \mathrm{cc}, \mathrm{ne}$ as inputs for the neural network. Each of these submatrices undergoes a dimensionality expansion to $\widetilde{\boldsymbol{x}}_{\mathrm{pa}} \in \mathbb{R}^{M\times 2\times 1}, \widetilde{\boldsymbol{x}}_{\mathrm{cc}} \in \mathbb{R}^{M\times|\mathrm{cc}|\times 1}, \widetilde{\boldsymbol{x}}_{\mathrm{ne}} \in \mathbb{R}^{M\times|\mathrm{ne}|\times 1}$. Next, three separate feed-forward subnetworks $l_{\mathrm{pa}}, l_{\mathrm{cc}}, l_{\mathrm{ne}}$ are applied to the three inputs $\boldsymbol{x}_{\mathrm{pa}}, \boldsymbol{x}_{\mathrm{cc}}, \boldsymbol{x}_{\mathrm{ne}}$. Each of these layers has the same architecture: For an input $\widetilde{\boldsymbol{x}} \in \mathbb{R}^{M\times d\times 1}$, weight matrices $\boldsymbol{W}_1 \in \mathbb{R}^{1\times C}$ and $\widetilde{\boldsymbol{W}}_1 \in \mathbb{R}^{C\times C}$, together with a bias vector $\boldsymbol{b}_1 \in \mathbb{R}^C$ are used to embed $\widetilde{\boldsymbol{x}}$ to

$$\boldsymbol{h}_1 = \tanh(\widetilde{\boldsymbol{x}}\boldsymbol{W}_1 + \boldsymbol{b}_1)\widetilde{\boldsymbol{W}}_1 \in \mathbb{R}^{M\times d\times C}$$

Then, two residual layers of the form

$$\boldsymbol{h}_{i+1} = \boldsymbol{h}_i + \tanh(\boldsymbol{h}_i\boldsymbol{W}_i + \boldsymbol{b}_i)\widetilde{\boldsymbol{W}}_i$$

with $\boldsymbol{W}_i \in \mathbb{R}^{C\times C}$, $\widetilde{\boldsymbol{W}}_i \in R^{C\times C}$, $\boldsymbol{b}_i \in \mathbb{R}^C$, $i = 1, 2$ are applied to obtain representations $\boldsymbol{h}_{\mathrm{pa}} \in \mathbb{R}^{M\times 2\times C}, \boldsymbol{h}_{\mathrm{cc}} \in \mathbb{R}^{M\times|\mathrm{cc}|\times C}, \boldsymbol{h}_{\mathrm{ne}} \in \mathbb{R}^{M\times|\mathrm{ne}|\times C}$. For the addition of the bias terms, broadcasting is used, i.e.,

$$\boldsymbol{b} \in \mathbb{R}^C \mapsto \widetilde{\boldsymbol{b}} \in \mathbb{R}^{M\times d\times C} \text{ with } \widetilde{\boldsymbol{b}}_{m,l,c} = \boldsymbol{b}_c \quad m = 1, \dots, M,\ l = 1 \dots, d,\ c = 1, \dots, C.$$

Now, the representations are concatenated along the dimension axis to obtain a $M \times (2 + |\mathrm{cc}| + |\mathrm{ne}|) \times C$ tensor. We use $d = 2 + |\mathrm{cc}| + |\mathrm{ne}|$. We employ the same observation-to-dependency network as before, but instead of using softmax on the output of the LogEig-Layer, we use mean-pooling

$$\boldsymbol{X} \mapsto \left(\frac{1}{d}\boldsymbol{X}^{T(3,2,1)}\mathbf{1}_d\right)^T$$

to inflate one of the variable axes of the $\mathbb{R}^{d\times d\times C}$ output of LogEig to obtain a $\mathbb{R}^{d\times C}$ batch of C vectors of dimension d. After applying a dense $C \times 1$ layer, we obtain a vector of length d. Now, the entries corresponding to the potential common children can be sliced out and backpropagated for training.

The network architecture is shown in figure 1.

D PARAMETERIZATION OF THE SEM

D.1 Chebyshev Polynomials for Training

For the Chebyshev polynomial, we utilize the following parameterization:

$$\begin{aligned} f_v(\boldsymbol{x}_{\mathrm{pa}}, \epsilon_1, \epsilon_2) = &\sum_{w \in \mathrm{pa}_{\mathcal{G}}(v)} \beta_w \sum_{n=1}^{r} \alpha_n T_n(x_w) + \epsilon_1 \\ &+ \alpha_{\mathrm{m}} \left(\sum_{s,t \in \mathrm{pa}(\mathcal{G}), s<t} \delta_{s,t} T_{\mathrm{m}}(x_s, x_t) + \sum_{w \in \mathrm{pa}(\mathcal{G})} T_{\mathrm{m}}(x_w, \epsilon_2) \right) \quad \forall v \in \mathcal{V}, \end{aligned}$$

where $r = 5$ is the degree, and T_n denotes the Chebyshev polynomials of the first kind (scaled for the input to have a maximum absolute value of 1), and bivariate polynomials are given by:

$$T_{\mathrm{m}}(x, y) := \frac{(x - \mu_x)(y - \mu_y)}{(1 + |\mu_x|)(1 + |\mu_y|)} \tag{1}$$

where m in the index stands for "multidimensional".

Here, $\mu_x \sim U[-1, 1]$, $\mu_y \sim U[-1, 1]$, and the coefficients $\alpha_1, \dots, \alpha_r, \alpha_{\mathrm{m}}$ are calculated from $\alpha_i = \frac{\widetilde{\alpha}_i}{\sum_n \widetilde{\alpha}_n + \widetilde{\alpha}_{\mathrm{m}}}$ for $i = 1, \dots, r$, with $\widetilde{\alpha}_i = \frac{\gamma_i}{i!}$, $\gamma_i \sim U[-1, 1]$, and $\widetilde{\alpha}_{\mathrm{m}} \sim U[-1, 1]$. $\beta_w = \frac{\widetilde{\beta}_w}{|\mathrm{pa}|}$ with $\widetilde{\beta}_w \sim U[0.7, 1.3]$. $\delta_{s,t}$ are random weights with $\delta_{s,t} = \frac{\widetilde{\delta}_{s,t}}{\sum_{s,t \in \mathrm{pa}(\mathcal{G}), s<t} |\widetilde{\delta}_{s,t}|}$, $\widetilde{\delta}_{s,t} \sim U[-1, 1]$

This parameterization is motivated by the observation that for a smooth function—where higher-order derivatives are not significantly larger than the lower-order ones—the coefficients of the Chebyshev approximation decrease in a factorial manner, as noted by Xiang and Liu (2020). Rapidly decreasing Chebyshev coefficients were also observed by Trefethen (2008). This provides a rationale for training the neural network on 'typical' smooth functional dependencies. Furthermore, this suggests that using Chebyshev polynomials of degree $r = 5$ is not a significant limitation, as the coefficients of higher orders are already negligibly small.

While multivariate Chebyshev polynomials constructed using Weyl-Groups were considered in (Hoffman and Withers, 1988; Puschel and Rotteler, 2007), we argue that our multivariate terms behave more nicely since they are bounded between $[-1, 1]$, akin to univariate Chebychev polynomials. Their construction also intuitively incorporates multiplicative effects of two variables in a randomly shifted way.

In order to prevent the values from exploding and to properly account for the common domain of Chebyshev polynomials, we implement several measures. Firstly, we scale each input to the SEM by the maximum value within the batch. Secondly, we standardize all variables; we subtract the mean and divide by the standard deviation for each batch. This ensures the variables are both centered and scaled. To further improve stability and robustness of our model, we introduce thresholds for any absolute values exceeding 5, thereby mitigating the potential impact of outliers.

D.2 Gaussian Mixture Error Terms

The additive error term ϵ_1 follows a Gaussian mixture distribution. We randomly determine the number of components $L \sim U\{1, \dots, 5\}$ from a discrete uniform distribution. Each component has randomly assigned parameters for the means $\mu_l \sim U[-1, 1]$, standard deviations $\sigma_l \sim U[0.05, 1]$, and weights $\widetilde{w}_l \sim U[0.3, 1]$, $w_l = \frac{\widetilde{w}_l}{\sum_l \widetilde{w}_l}$ such that $\epsilon_1 \sim \sum_{l=1}^{L} w_l N(\mu_l, \sigma_l^2)$. The multiplicative error term ϵ_2 is uniformly distributed, with $\epsilon_2 \sim U[-1, 1]$.

D.3 Testing Dependencies

To create testing data, we create synthetic data according to an SEM equipped with different dependency function, while the error term follows a Gaussian mixture distribution as before. We use the following dependencies for testing: Chebyshev, linear, sine, cosine, x^2, x^3, multidimensional multiplicative dependency. The Chebyshev-dependency used was the same as in the training procedure. We used the following testing dependency functions: x, $\sin(x)$, $\cos(x)$, x^2, x^3 as $g(x)$ in

$$f_{\text{test}}(\boldsymbol{x}_{\text{pa}}, \varepsilon) = \sum_{w \in \text{pa}(\mathcal{G})} \alpha_w g(x_w) + \epsilon.$$

For the multi-dimensional multiplicative test dependency, we used

$$f_v(\boldsymbol{x}_{\text{pa}}, \epsilon_1, \epsilon_2) = \alpha_{\text{m}} \left(\sum_{s,t \in \text{pa}(\mathcal{G}), s<t} \delta_{s,t} T_{\text{m}}(x_s, x_t) + \sum_{w \in \text{pa}(\mathcal{G})} T_{\text{m}}(x_w, \epsilon_2) \right) \quad \forall v \in \mathcal{V},$$

with

$$T_{\text{m}}(x, y) := \frac{(x - \mu_x)(y - \mu_y)}{(1 + |\mu_x|)(1 + |\mu_y|)}$$

with $\mu_x \sim U[-1,1]$, $\mu_y \sim U[-1,1]$, $\delta_{s,t} = \frac{\widetilde{\delta}_{s,t}}{\sum_{s,t \in \text{pa}(\mathcal{G}), s<t} |\widetilde{\delta}_{s,t}|}$, $\widetilde{\delta}_{s,t} \sim U[-1,1]$.

E TRAINING DETAILS

E.1 Model Hyperparameters

The implementation was performed using TensorFlow (Abadi et al., 2015). We employed the ADAM optimizer by Kingma and Ba (2015). For training, we used the hyperparameters stated in Table 1:

Table 1: Hyperparameters for the undirected graph estimation

Hyperparameter	Value
Layer Parameters	
Number of channels C	100
Number of inner channels c	100
Maximal degree activation function	3
Attention heads	5
Number of layers	
Attention between attributes	10
Attention between samples	10
$C \times C$ dense observational layers	10
bilinear attention + SPD activation	10
Training Schedule	
epochs	1000
samples per epoch	128
Initial learning rate	0.0005
Learning rate decrease factor	$\left(\frac{1}{10}\right)^{1/500}$
Minibatchsize	1

Table 2: Hyperparameters of the CPDAG estimation model

Hyperparameter	Value
Layer Parameters	
Number of channels C	100
Number of inner channels c	100
Maximal degree activation function	3
Attention heads	5
Number of layers	
Attention between attributes	10
Attention between samples	10
$C \times C$ dense observational layers	10
bilinear attention + SPD activation	10
Training Schedule	
epochs	1000
matrices per epoch	1
Initial learning rate	0.0005
Learning rate decrease factor	$\left(\frac{1}{10}\right)^{1/1000}$
Minibatchsize	1

Additionally, we generated data with a random number of samples $M \sim U\{50, 51, \dots, 1000\}$ and a random variable dimension $d \sim U\{10, 11, \dots, 100\}$.

Ablation Study In the ablation study, we evaluate the performance of the full model in comparison to models with reduced complexities. The full model is comprised of two attention between attributes layers, two attention between samples layers, two dense layers, and four bilinear layers equipped with SPD activation functions. This setup maintains parity between the number of attention layers operating on observational data and those focusing on covariance data, while also ensuring a comparable parameter count across different configurations. All models in the study utilize $C = c = 100$ channels and are trained over 500 epochs, with each epoch comprising 128 data matrix / adjacency label pairs. Again, we generated data with a random number of samples $M \sim U\{50, 51, \dots, 1000\}$ and a random variable dimension $d \sim U\{10, 11, \dots, 100\}$.

E.2 Loss Function for the Three-Class Edge Classification Problem.

We employ the categorical cross-entropy loss function for classifying edges into one of three categories: *no-edge*, *skeleton edge*, and *moralized edge*. Additionally, to enforce the condition that a moral edge between nodes X and Y should only be predicted if there is a potential common child Z (i.e., $X - Z$ and $Z - Y$), we introduce a penalty term, $\mathcal{L}_\mathrm{p}$.

The overall loss function is defined as $\mathcal{L}_\mathrm{b} + \mathcal{L}_\mathrm{c} + \mathcal{L}_\mathrm{p}$. Here, $\mathcal{L}_\mathrm{c}$ is the categorical crossentropy of the three categories given by

$$\mathcal{L}_\mathrm{c}(\boldsymbol{A}, \widehat{\boldsymbol{A}}) := H(\boldsymbol{A}, \widehat{\boldsymbol{A}}) = -\sum_{i=2}^{d} \sum_{j=1}^{i-1} \sum_{c=1}^{3} A_{i,j,c} \log(\widehat{A}_{i,j,c})$$

denotes the categorical crossentropy of the three categories *no-edge*, *skeleton edge*, and *moralized edge* with

$$\boldsymbol{A} \in \mathbb{R}^{d \times d \times 3} \quad \text{with} \quad A_{i,j,c} = \begin{cases} 1 & \text{if } (i,j) \text{ is in category } c \text{ in the ground-truth DAG} \\ 0 & \text{else} \end{cases}$$

and $\widehat{A}_{i,j,c}$ denotes the estimation by the algorithm on it. $\boldsymbol{A}$ and $\widehat{\boldsymbol{A}}$ are symmetric along its first two axes, i.e., $A_{i,j,c} = A_{j,i,c}, \quad i,j = 1,\dots,d, \quad c = 1,\dots,3.$

$\mathcal{L}_\text{b}$ denotes the binary loss of no-edge vs. any edge present (present edges = skeleton edges $\cup$ moralized edges):

$$\mathcal{L}_\text{b}(\boldsymbol{A}^{(b)}, \widehat{\boldsymbol{A}}^{(b)}) := H(\boldsymbol{A}^{(b)}, \widehat{\boldsymbol{A}}^{(b)}) = -\sum_{i=2}^{d}\sum_{j=1}^{i}\left[A_{i,j}^{(b)}\log(\widehat{A}_{i,j}^{(b)}) + (1 - A_{i,j}^{(b)})\log(1 - \widehat{A}_{i,j}^{(b)})\right]$$

with

$$\boldsymbol{A}^{(b)} \in \mathbb{R}^{d \times d} \quad \text{with} \quad A_{i,j}^{(b)} = \begin{cases} 1 & \text{if an edge is estimated between } i \text{ and } j \\ 0 & \text{else} \end{cases}$$

being the adjacency matrix of no-edge vs. (direct edge $\cup$ moralized edge).

The penalty term, $\mathcal{L}_\text{p}$ is defined as:

$$\mathcal{L}_\text{p}(\widehat{\boldsymbol{A}}) := \max\left(\widehat{\boldsymbol{A}}_3 - [\widehat{\boldsymbol{A}}_2\widehat{\boldsymbol{A}}_2]^{0.5}, 0\right).$$

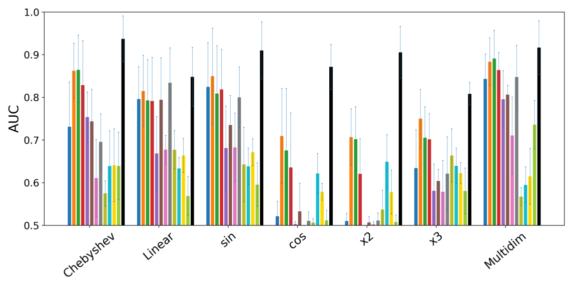
$\widehat{\boldsymbol{A}}_1, \widehat{\boldsymbol{A}}_2, \widehat{\boldsymbol{A}}_3 \in \mathbb{R}^{d \times d}$ are estimates of $\boldsymbol{A}_{\cdot,\cdot,1}$, $\boldsymbol{A}_{\cdot,\cdot,2}$, and $\boldsymbol{A}_{\cdot,\cdot,3}$ by the algorithm respectively. This term penalizes the prediction of a moralized edge in the absence of potential common child edges. The square root operation is applied element-wise.

F FURTHER EXPERIMENTS

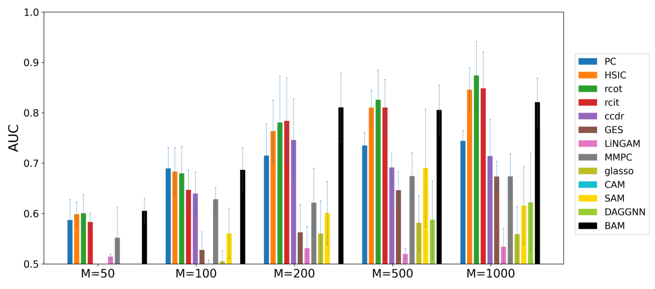
F.1 Error Bars

In our experiments, each algorithm was evaluated on five distinct trials, each involving a unique data matrix and corresponding ground-truth graph. The bars in the figures represent the mean performance values across these trials, while the error bars indicate the standard deviation. It is important to note that the observed variability, manifested as relatively large error bars, is predominantly due to the random sampling of graph degrees, which has a substantial influence on estimation accuracy. Despite this inherent variability, the comparison across algorithms remains valid, as each algorithm is tested under the same conditions concerning underlying graph density. Therefore, the magnitude of the error bars should not be interpreted as undermining the reliability of our findings.

F.2 Additional Results on Undirected Graph Estimation



(a)



(b)

Figure 2: AUC values for undirected graph estimation: (a) in the low-dimensional setting $d = 20$, $M = 500$ (b) under MLP-based dependency.

Figure 2 provides supplemental data on the task of undirected graph estimation. Panel (a) showcases performance in a low-dimensional setting characterized by $d = 20$ and $M = 500$. In this scenario, our method (BAM) consistently outperforms competing graph inference algorithms. To further assess its capability to recognize multidimensional dependencies, we extended our tests to cases where the dependency function within the SEM is modeled via a randomly initialized multilayer perceptron (MLP) with a random number of layers $\sim \mathcal{U}\{1, \ldots, 5\}$, a random number of hidden layers $\sim \mathcal{U}\{4, 64\}$, and relu or tanh activation with probability 0.5 each. Despite these complexities, our algorithm maintained state-of-the-art performance, delivering AUC scores comparable to the top-performing existing methods such as HSIC, rcot, and rcit, as depicted in Panel 2 (b).

F.3 Additional Results on CPDAG Estimation

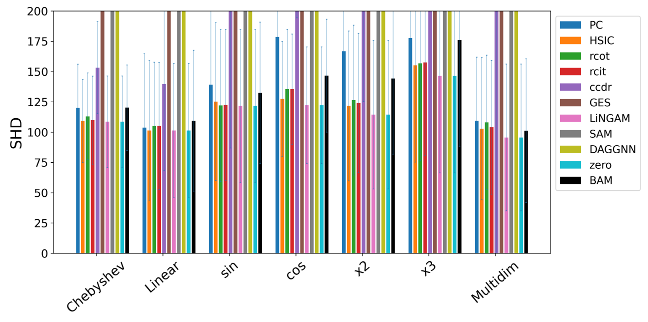


Figure 3: SHD values for the high-dimensional CPDAG estimation $d = 100$, $M = 50$. "zero" denotes the zero graph without any edges.

CPDAG estimation in high-dimensional settings presents significant challenges. In the specific case of $d = 100$ and $M = 50$, none of the algorithms we evaluated could outperform a zero-graph (i.e., a graph with no edges) baseline in terms of Structural Hamming Distance (SHD). The results, depicted in Figure 3, substantiate this observation and suggest that CPDAG estimation remains a difficult problem under these conditions, at least with our chosen graph density setup. Given these challenges in high-dimensional settings, we prioritize evaluation based on AUC over SHD. This underscores the utility of undirected graph methods for such problems, as directed approaches may not only be inefficient but also risk yielding misleading interpretations.

F.4 Time Comparison

Figure 4 depicts the average runtimes per evaluation step, accompanied by their corresponding standard deviations. Specifically, the results are presented in the form of mean ± one standard deviation. The x-axis enumerates various sample sizes, denoted as M, while both mean and standard deviation were computed based on 5 independent inference tests for each configuration with a fixed sample size M and graph dimension d.

These empirical observations substantiate the computational efficiency of our algorithm in the inference phase. Notably, the algorithm avoids the necessity for re-training when confronted with varying sample sizes M or dimensions d, thereby demonstrating its robustness and scalability.

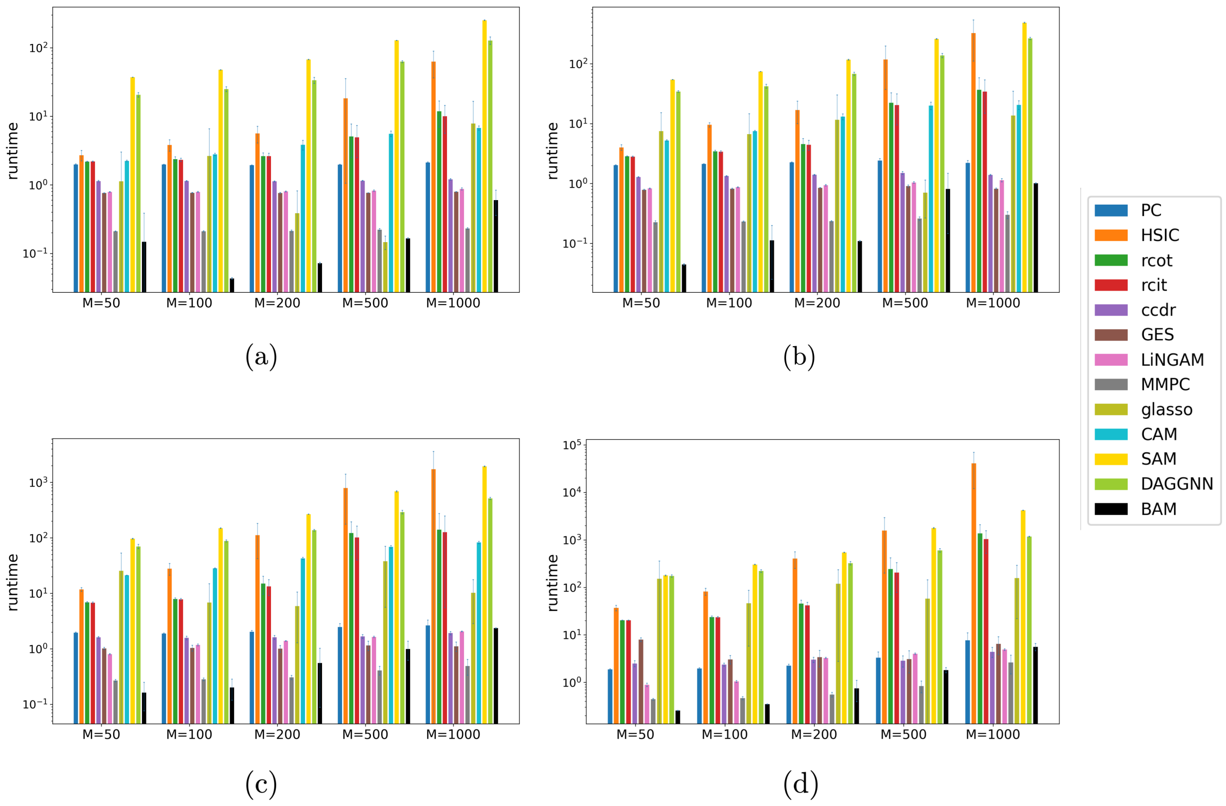


Figure 4: Algorithm runtime for (a) $d = 10$, (b) $d = 20$, (c) $d = 50$, (d) $d = 100$ in seconds per $M \times d$ data matrix inference.

G INTUITION

G.1 Shape-Agnostic Architecture and the Role of Attention Layers

When employing a shape-agnostic architecture for matrices $\in \mathbb{R}^{M\times d}$, it is crucial to ensure that all elements within the $M \times d$ matrix can interact and influence one another. Consider a scenario where one axis of the matrix is expanded to the shape $\mathbb{R}^{M\times d\times 1}$, followed by dense layers with $1 \times C$ and $C \times C$ weights. In this configuration, the dense layers carry out element-wise operations on the $M \times d$ elements, processing them in isolation from each other. This is because each hidden representation is essentially a linear combination of C matrices of shape $M \times d$ prior to the application of an element-wise activation function.

This limitation is addressed by incorporating attention layers into the architecture. These layers adaptively compute non-trainable $M \times M$ and $d \times d$ attention matrices based on trainable $C \times C$ weights. This approach allows for a shape-agnostic architecture, as the same set of trainable weights can be employed for any $M \times d$ matrix, while still enabling the matrix entries to influence each other. In this way, the attention mechanism becomes an essential component of our model. Although we also experimented with Network-in-Network methods (Lin et al., 2013), we found that the attention mechanism offers a more stable, efficient, and straightforward computation of non-trainable $M \times M$ and $d \times d$ attention matrices, using only trainable $C \times C$ weights.

Our proposed bilinear attention mechanism is, to our knowledge, the first SPD layer to enable shape-agnostic computations. It uses trainable $C \times C$ weights to calculate non-trainable $d \times d$ attention matrices, allowing for adaptive weighting across different $d \times d$ SPD matrix sizes. This flexibility makes it a unique and essential component of our architecture. Additionally, this construction ensures the desired permutation invariance among the variables. Our approach essentially learns matrix operations that should be applicable to any input matrix with arbitrary $M \times d$ input.

G.2 Attention scores in the BAM layer

Consider the setting as in Figure 3 (right) and the computation of the output $\boldsymbol{H}$ by $\boldsymbol{A} \otimes \boldsymbol{S}$, where $\boldsymbol{A} \in \mathcal{S}_{\succeq}^{d\times d\times C}$ are the attention scores and $\boldsymbol{S} \in \mathcal{S}_{\succeq}^{d\times d\times C}$ are the input matrices into the BAM layer. Since $\boldsymbol{A} \otimes \boldsymbol{S}$ is processed parallel across the channels, we consider for simplicity the output of a single channel here and assume $\boldsymbol{A} \in \mathcal{S}_{\succeq}^{d\times d}$ and $\boldsymbol{S} \in \mathcal{S}_{\succeq}^{d\times d}$ to be quadratic, positive definite $d \times d$ matrices.

While traditional self-attention computes scores to assess the importance of one data point to another, our bilinear attention mechanism extends this by exploring the interdependence of variable pairs. Specifically, for an output pair (i, j), its associated output value is determined not merely by a direct scalar relationship but by the bilinear form: $\sum_{k,l} A_{i,k} S_{k,l} A_{l,j}$. Thus, instead of a singular focus on the relation "How does j affect i?", quantified in the score matrix $\boldsymbol{A}$ in classical attention, the score matrix in bilinear attention shows the interaction strengths of pair sets $\{(i, k)|k = 1, \dots, d\}$ and $\{(l, j)|l = 1, \dots, d\}$. The "receptive field" adopts a cross-form within the scores $\boldsymbol{A}$ instead of being $A_{i,j}$ only, in the sense that relevant scores for the output at position (i, j) are not limited to $A_{i,j}$ but $\{\boldsymbol{A}_{i,\cdot} \cup \boldsymbol{A}_{\cdot,j}\}$.

G.3 Keys and Queries

Continuing with the single-channel assumption due to parallel channel processing, consider the quadratic form $\mathcal{S}^{d\times d} \times \mathcal{S}^{d\times d} \mapsto \mathcal{S}^{d\times d}$, $(\boldsymbol{K}, \boldsymbol{Q}) \mapsto \boldsymbol{K}^T\boldsymbol{Q}\boldsymbol{K}$ of the key-query interaction. The (i, j)-th entry of $\boldsymbol{K}^T\boldsymbol{Q}\boldsymbol{K}$ is $\boldsymbol{K}_i^T\boldsymbol{Q}\boldsymbol{K}_j$ for the columns $\boldsymbol{K}_1, \dots, \boldsymbol{K}_d$ of $\boldsymbol{K}$, which are often referred to as keys. Using the eigendecomposition of $\boldsymbol{Q} = \boldsymbol{U}^T\boldsymbol{D}\boldsymbol{U}$ one obtains for the (i, j)-th entry the bilinear form $(\boldsymbol{U}\boldsymbol{K}_i)^T\boldsymbol{D}(\boldsymbol{U}\boldsymbol{K}_j)$. Note that $\boldsymbol{U}\boldsymbol{K}_i$ is a similarity measure between $\boldsymbol{K}$ and $\boldsymbol{Q}$ analogous to standard attention. So, for bilinear attention, similarity scores are calculated between the keys $\boldsymbol{K}$ and the eigenvectors of the queries $\boldsymbol{Q}$. Afterwards, the $\boldsymbol{D}^{\frac{1}{2}}$-weighted bilinear-form $(\boldsymbol{D}^{\frac{1}{2}}\boldsymbol{U}\boldsymbol{K}_i)^T(\boldsymbol{D}^{\frac{1}{2}}\boldsymbol{U}\boldsymbol{K}_j)$ is used to create covariance matrices by combining the similarity scores between $\boldsymbol{U}$ and $\boldsymbol{K}$. Hence, in bilinear attention, the similarity scores are functions of both the i-th and j-th keys as well as all queries. This is consistent with the attention-score behavior, where the interaction strengths of all pair sets $\{(i, k)|k = 1, \dots, d\}$ and $\{(l, j)|l = 1, \dots, d\}$ collectively influence the output.

This is in contrast to standard attention, which uses the untransformed dot product $(\boldsymbol{k}, \boldsymbol{q}) \in \mathbb{R}^d \times \mathbb{R}^d \mapsto \boldsymbol{k}^T\boldsymbol{q} \in \mathbb{R}$ for the columns of key and query matrices $\boldsymbol{K}$, $\boldsymbol{Q}$.

H LIMITATIONS

The model effectively captures smooth dependence relations using Chebyshev polynomials. Although this approach excels across various types of dependencies, it might have limitations for data structures that deviate significantly from the generated synthetic data. However, adapting the synthetic data generation to accommodate these structures is straightforward.

As for the Log-Eig layer, it performs efficiently within a moderate dimensional range but may face computational challenges when scaling to higher dimensions. Training our model with parameters similar to those used in this study demands substantial memory resources; in our experiments, around 80 GB of GPU memory was required.

The attention mechanism, while effective, can be costly for high-dimensional (both, in M and d) inputs. This can be ameliorated with local attention, although this approach may introduce its own set of challenges. While the model is effective for its intended applications, its architecture allows for easy extensions. For instance, a separate embedding layer could be added for both observational and interventional data to make use of interventional data.

An end-to-end approach for CPDAG estimation might offer further benefits. The current model loses directional information in the covariance computation, making an end-to-end approach for CPDAG estimation unfeasible with the existing architecture. However, a simple extension could involve using two separate embeddings for each variable, one for the variable as parent, one for the variable as child, effectively doubling the dimensionality to $2d$, to potentially facilitate directional inference. Lastly, like other neural network-based approaches, there is a potential risk of overfitting (here on Chebyshev polynomial dependencies), necessitating hyperparameter tuning.

References

Abadi, M., Agarwal, A., Barham, P., and et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Apicella, A., Donnarumma, F., Isgrò, F., and Prevete, R. (2021). A survey on modern trainable activation functions. *Neural Networks*, 138:14–32.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer Normalization. *arXiv:1607.06450*.

Bachlechner, T., Majumder, B. P., Mao, H., Cottrell, G., and McAuley, J. (2021). ReZero is All You Need: Fast Convergence at Large Depth. In *UAI*, pages 1352–1361.

Chung, H., Lee, S. J., and Park, J. G. (2016). Deep Neural Network Using Trainable Activation Functions. In *IJCNN*, pages 348–352.

Guillot, D. and Rajaratnam, B. (2015). Functions Preserving Positive Definiteness for Sparse Matrices. *Proc. Am. Math. Soc.*, 367:627–649.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *CVPR*, pages 770–778.

Hoffman, M. E. and Withers, W. D. (1988). Generalized chebyshev polynomials associated with affine weyl groups. *Transactions of the American Mathematical Society*, 308(1):91–104.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques* . MIT press.

Lin, M., Chen, Q., and Yan, S. (2013). Network in network. *arXiv preprint arXiv:1312.4400*.

Puschel, M. and Rotteler, M. (2007). Algebraic signal processing theory: 2-d spatial hexagonal lattice. *IEEE Transactions on Image Processing*, 16(6):1506–1521.

Schoenberg, I. J. (1942). Positive definite functions on spheres. *Duke Mathematical Journal*, 9:96–108. MR0005922 (3,232c).

Touvron, H., Cord, M., Sablayrolles, A., Synnaeve, G., and Jégou, H. (2021). Going deeper with Image Transformers. In *ICCV*, pages 32–42.

Trefethen, L. N. (2008). Is Gauss Quadrature Better than Clenshaw–Curtis? *SIAM Review*, 50(1):67–87.

Wu, Y. and He, K. (2018). Group Normalization. In *ECCV*, pages 3–19.

Xiang, S. and Liu, G. (2020). Optimal decay rates on the asymptotics of orthogonal polynomial expansions for functions of limited regularities. *Numerische Mathematik*, 145(1):117–148.

Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., et al. (2020). On Layer Normalization in the Transformer Architecture. In *ICML*, volume 119, pages 10524–10533.