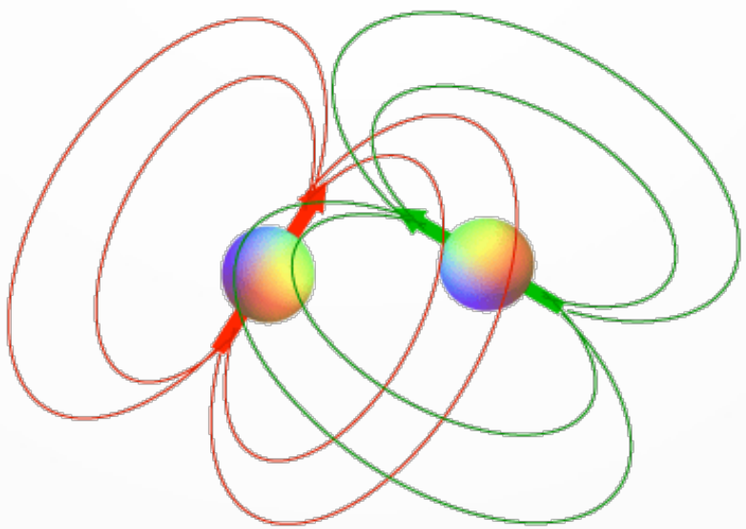
Predicting Molecular Properties



Pedro A. Campana

Intelligent Data Analysis II – Winter semester 21/22

Introduction

- NMR finds molecular structure based on interactions between atoms

- NMR depends on scalar-coupling prediction

- This is done using expensive quantum mechanics

- Objective: Predict scalar-coupling using molecular data.

Methods – available data

Competition:

- Training data: 85012 Molecules

- Test data: Disjoint, 45777 Molecules

On this work:

- Training data: 90% of the original training data

- Test data: 10% hold-out data

- Same split used for all examples

Methods – available data

Target variable:

8 Different groups, depending on distance and atom types:

3JHC	1511207
2JHC	1140867
1JHC	709133
3JHH	590529
2JHH	377988
3JHN	166613
2JHN	119059
1JHN	43680

Only 1JHN, 2JHH, 1JHN and 2JHN modeled in this work

Methods – available data

Data represented as multiplex graphs:

Node features:
- Different physical properties of the atoms.
- Measured properties for this molecule (mulliken charges, magnetic shielding tensors)

Graph features (added as node features):
- Dipole moment
- Potential energy

Edge features (resulting in different Adjacency matrices):
- Distance between pair
- Cos of the angle formed with the dipole moment
- Lowest knn-neighborhood
- (Dihedrals for 2nn edges)

Methods – Models explored

Baseline model: Fully connected network, with residual connections and learnable gates.

Contending models: SOTA Attention-based GNNs for heterogeneous graph learning, followed by FC with residual connections:
- GATv2 from “How Attentive are Graph Attention Networks?”
- Pathfinder Discovery Networks from “Pathfinder Discovery Networks for Neural Message Passing”
- Transformer GNN from “Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification”

Evaluation metrics:
- MAE
- Pearson R (predicted, observed)
- Ln(MAE) (Used in the kaggle competition)

Methods – Attention based GNN

General idea:

$$\mathrm{h}_{i_{n+1}} = \sigma(\Theta_s h_{i_n} + \sum_{j \in N(i)} \alpha(e_{ij}, h_{i_n}, h_{j_n}, H_n)\phi(e_{ij}, h_{i_n}, h_{j_n}))$$

At each layer represent features of each node as a function of the features from previous layer and a function of the neighbor features, weighted via an attention mechanism.

Methods – GATv2

Calculate attention logit for each pair, via concatenation of node (and edge) features:

$$e\left(\boldsymbol{h}_i, \boldsymbol{h}_j\right) = \boldsymbol{a}^{\top} \operatorname{LeakyReLU}\left(\boldsymbol{W} \cdot \left[\boldsymbol{h}_i \| \boldsymbol{h}_j\right]\right)$$

Map to (0, 1) range using softmax over neighborhood:

$$\alpha_{ij} = \operatorname{softmax}_j\left(e\left(\boldsymbol{h}_i, \boldsymbol{h}_j\right)\right) = \frac{\exp\left(e\left(\boldsymbol{h}_i, \boldsymbol{h}_j\right)\right)}{\sum_{j' \in \mathcal{N}_i} \exp\left(e\left(\boldsymbol{h}_i, \boldsymbol{h}_{j'}\right)\right)}$$

Obtain features at next layer:

$$\boldsymbol{h}_i' = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \cdot \boldsymbol{W} \boldsymbol{h}_j\right)$$

Methods – Transformer GNN

$$q_{c,i}^{(l)} = W_{c,q}^{(l)} h_i^{(l)} + b_{c,q}^{(l)}$$

$$k_{c,j}^{(l)} = W_{c,k}^{(l)} h_j^{(l)} + b_{c,k}^{(l)}$$

$$e_{c,ij} = W_{c,e} e_{ij} + b_{c,e}$$

$$\alpha_{c,ij}^{(l)} = \frac{\langle q_{c,i}^{(l)}, k_{c,j}^{(l)} + e_{c,ij} \rangle}{\sum_{u \in \mathcal{N}(i)} \langle q_{c,i}^{(l)}, k_{c,u}^{(l)} + e_{c,iu} \rangle}$$

$$v_{c,j}^{(l)} = W_{c,v}^{(l)} h_j^{(l)} + b_{c,v}^{(l)}$$

$$\hat{h}_i^{(l+1)} = \Big\|_{c=1}^{C} \Big[\sum_{j \in \mathcal{N}(i)} \alpha_{c,ij}^{(l)} (v_{c,j}^{(l)} + e_{c,ij}) \Big]$$

- Query: Affine transformation of source features
- Keys: Affine transformation of neighbor features + edge features
- Attention: Scaled dot product of query and keys, weighted over neighborhood.

Methods – Pathway discovery networks

Apparently different mathematical formulation, leading to equivalent solution:

$$\widetilde{\boldsymbol{G}} = \sigma\left(\sum_{i=1}^{D} \beta_i \cdot \widetilde{\boldsymbol{A}}_i\right).$$

$$\widetilde{\mathbf{G}}_{l+1,1}; \ldots; \widetilde{\mathbf{G}}_{l+1,q} = f^l\left(\widetilde{\mathbf{G}}_{l,1}; \ldots; \widetilde{\mathbf{G}}_{l,p}\right).$$

$$\mathbf{Z} = \hat{\sigma}(\mathbf{D}_{\hat{\mathbf{G}}}^{-1/2} \hat{\mathbf{G}} \mathbf{D}_{\hat{\mathbf{G}}}^{-1/2} \mathbf{X}\mathbf{W})$$

- Learn new adjacency matrix as weighted sum of the different adjacency matrices (edge features) in a single pathfinder neuron.

- Combine different pathfinder neurons in pathfinder network.

- Obtain new node features, as function of Symmetrically normalized Laplacian of learned graph and node features.

Methods – Complete architectures

Pytorch and Pytorch geometric implementations were used for the different GNNs. Ray-tune for hyperparameter tuning. 4 different final strategies. Trained for 150 epochs :

1. Generate embeddings via stacked GNN, connected through gated residual connections. After embedding, collapse them in one fully connected residual network for each class. Output scalar.

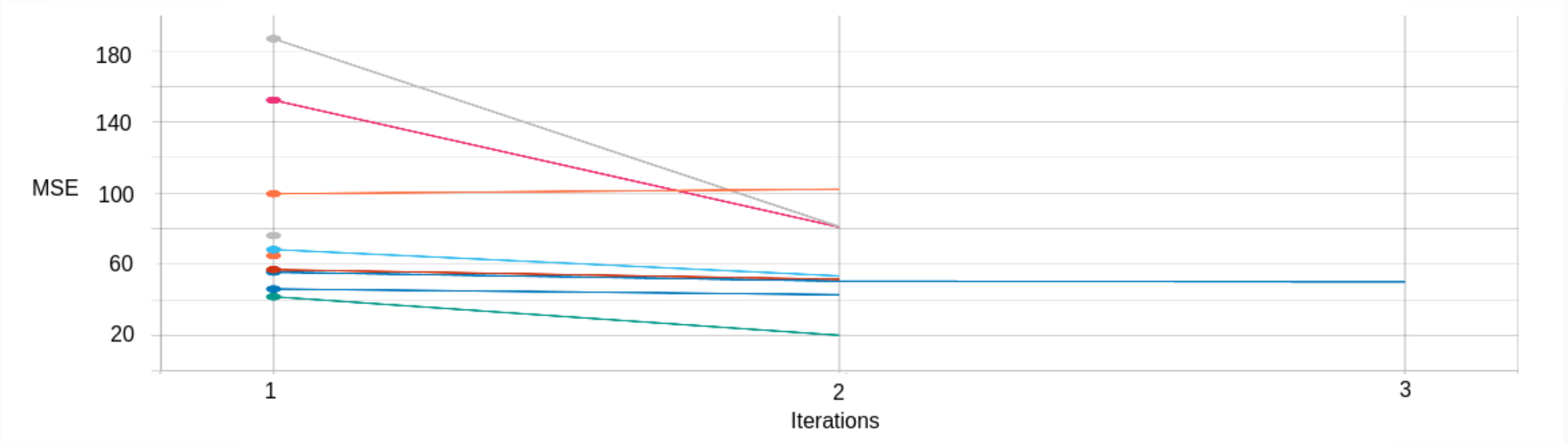
2. For each different target class, concatenate and tabularize embeddings resulting from converged (1.) top 3 models, collapse them to single scalar via fully connected residual network.

3. Fine-tune (1.) for a single target-class.

4. New features and BOHB hyperparameter optimization

Methods – BOHB Results

batch_acc	conv_features	dropout	lr	n_heads	n_layers	n_res	weight_decay	iter	total time (s)	loss
8	128	0.236607	0.0289597	3	4	3	1.25207e-08	1	871.553	187.151
2	64	0.146045	0.0129609	2	3	3	0.0287105	1	793.769	82.0989
8	128	0.0281704	0.0897016	4	4	3	1.25327e-05	1	880.737	5.20438e+22
2	128	0.389765	0.0422166	3	2	3	0.000175296	1	867.783	72.4392
4	128	0.00180811	0.000182754	3	3	4	1.43521e-08	5	4424.82	27.1775
2	64	0.347476	0.00138341	4	2	4	0.000270413	1	816.237	76.1351
4	128	0.105616	0.000179854	4	4	2	0.0217407	1	845.339	64.8294
8	64	0.1941	0.000908942	2	3	3	2.23318e-05	3	2373.61	50.0771
2	32	0.104384	0.000203085	4	3	2	1.50834e-07	2	1508.08	51.5369
4	128	0.100621	1.93067e-05	3	2	4	1.1883e-08	2	1747.74	53.447
8	64	0.338625	0.083435	3	4	2	0.00107739	2	1527.45	80.8971
2	128	0.0343089	0.000782569	3	3	3	1.76755e-08	2	1699.04	20.1051
2	128	0.391413	0.098462	4	4	3	0.0289604	2	1728.64	81.1686
4	32	0.0101121	0.00438898	4	3	3	0.0793442	2	1523.22	102.253
2	32	0.275541	0.00102496	2	2	2	1.00994e-05	2	1490.71	42.873

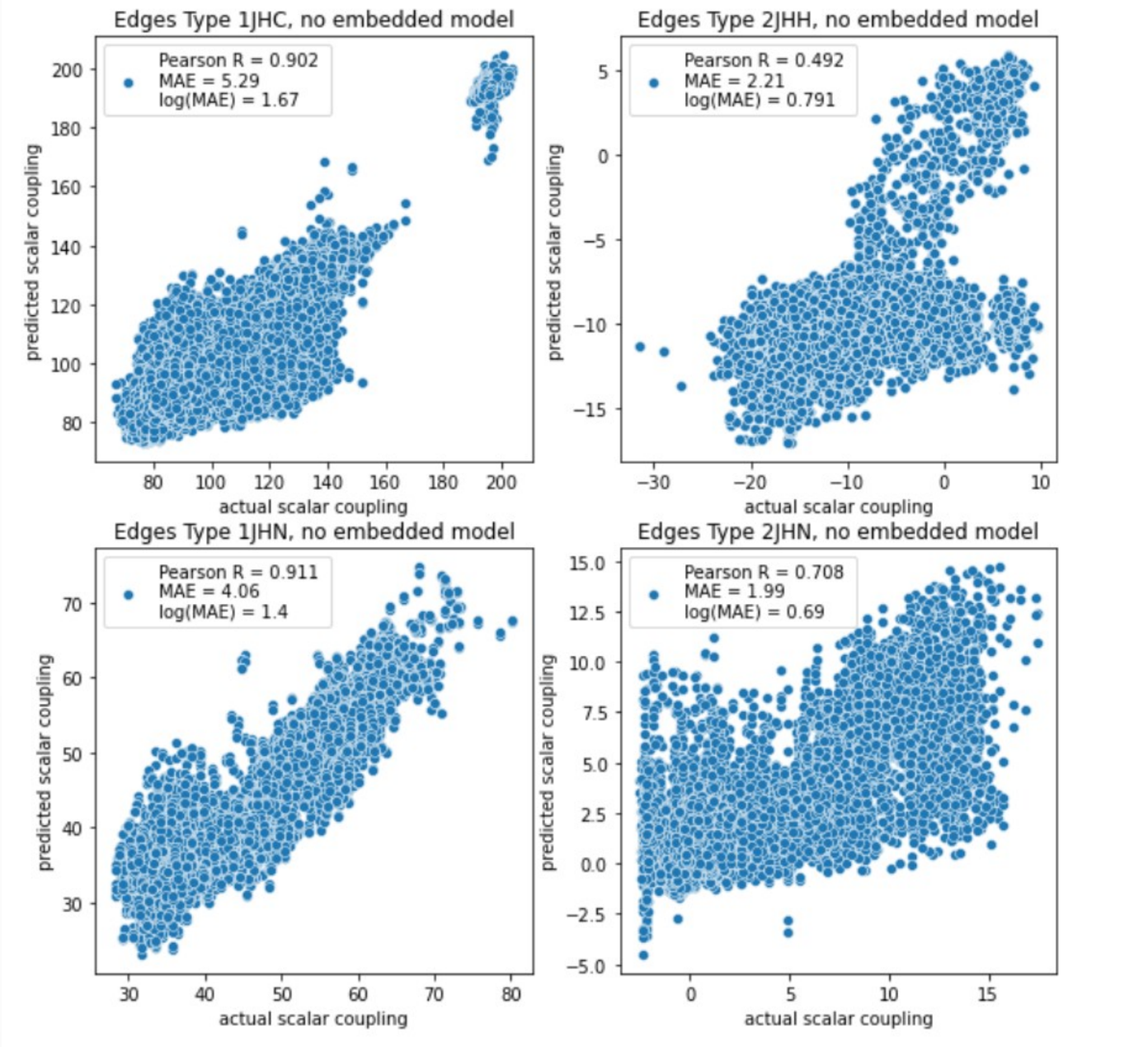


Methods – Retained models

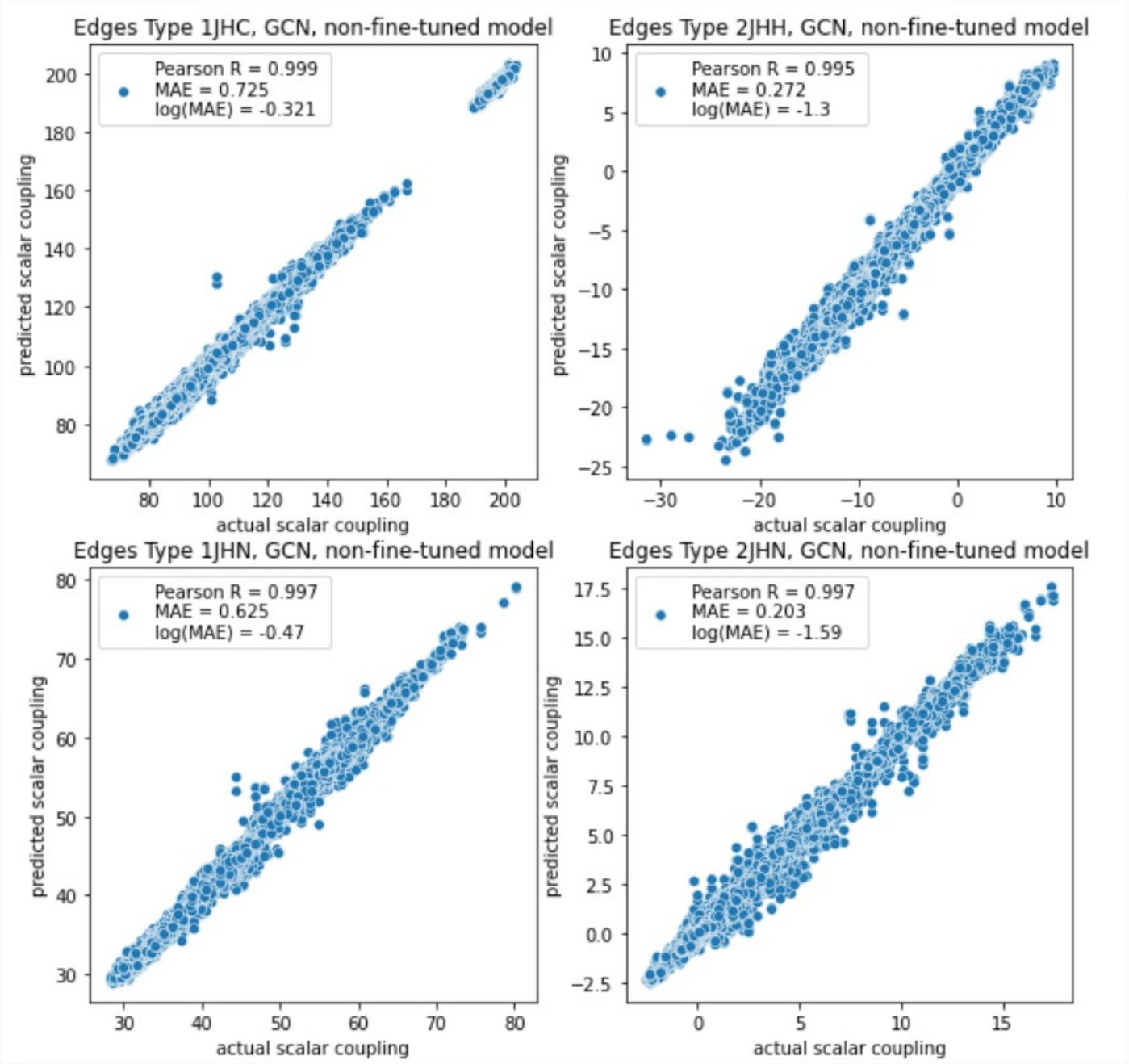
Label	Weight decay	Learning rate	Batch size	Batch acc.	Dropout	Convolutional layer	N conv layers	N of att. heads
GATv2	0.001	1E-06	128	None	0.001	GATv2	3	6
Transformer	0.001	1E-06	128	None	0.001	TransformerConv	3	6
PDN	0.001	1E-06	128	None	0.001	PDN	3	None
Fine-tuned	0.001	1E-06	128	None	0.001	GATv2	3	6
Tabularized	1E-05	0.0001	9192	None	0.05	None	None	None
BOHB	1.77E-08	0.00078	128	2	0.034	GATv2	3	3

N res. blocks	Features
2	Graph + Nodes + edges
2	Graph + Nodes + edges
2	Graph + Nodes + edges
2	Graph + Nodes + edges
2	Embeddings GATv2, Transformer, PDN
3	Graph + Nodes + edges (added dihedrals)

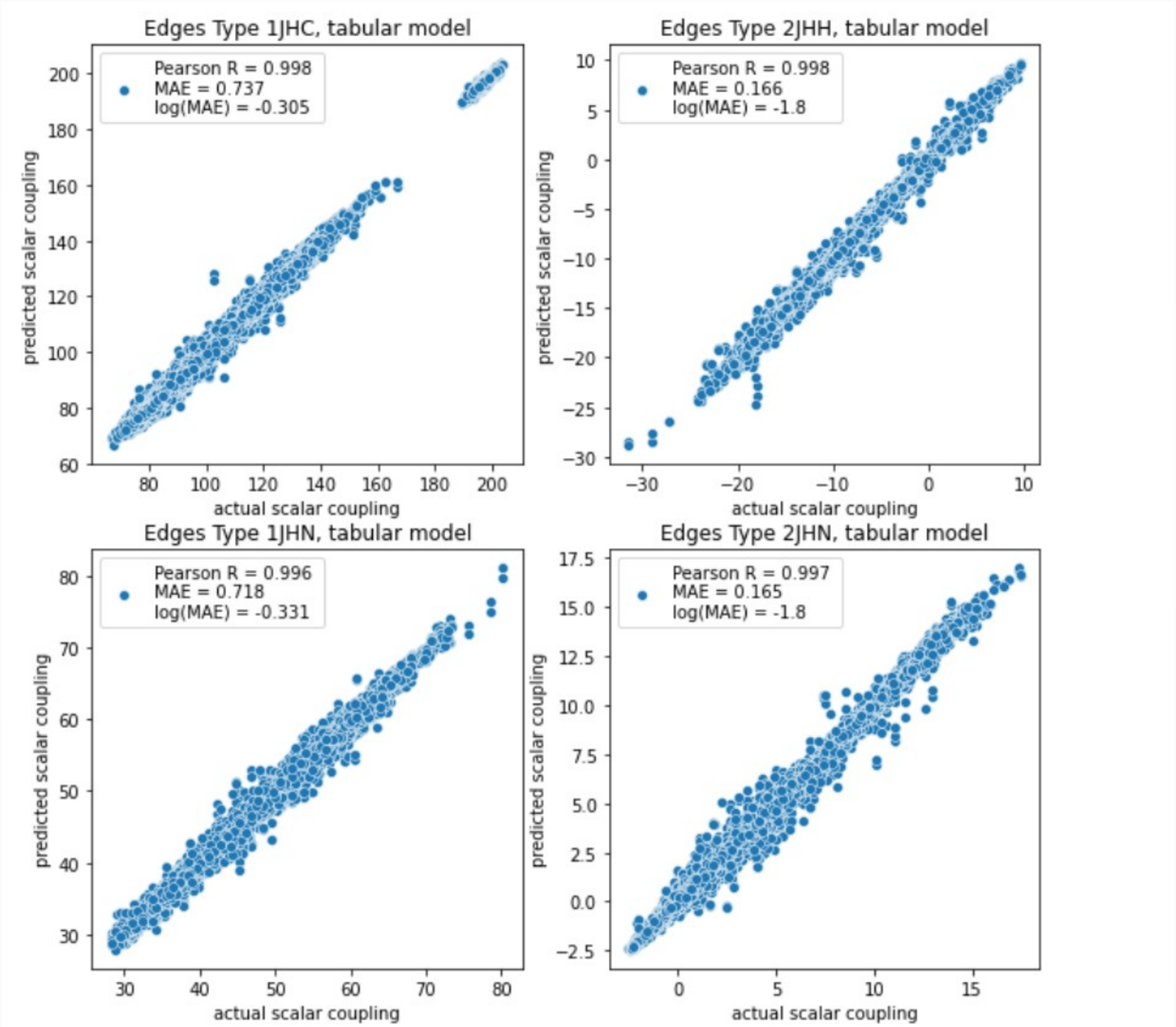
Results - Baseline



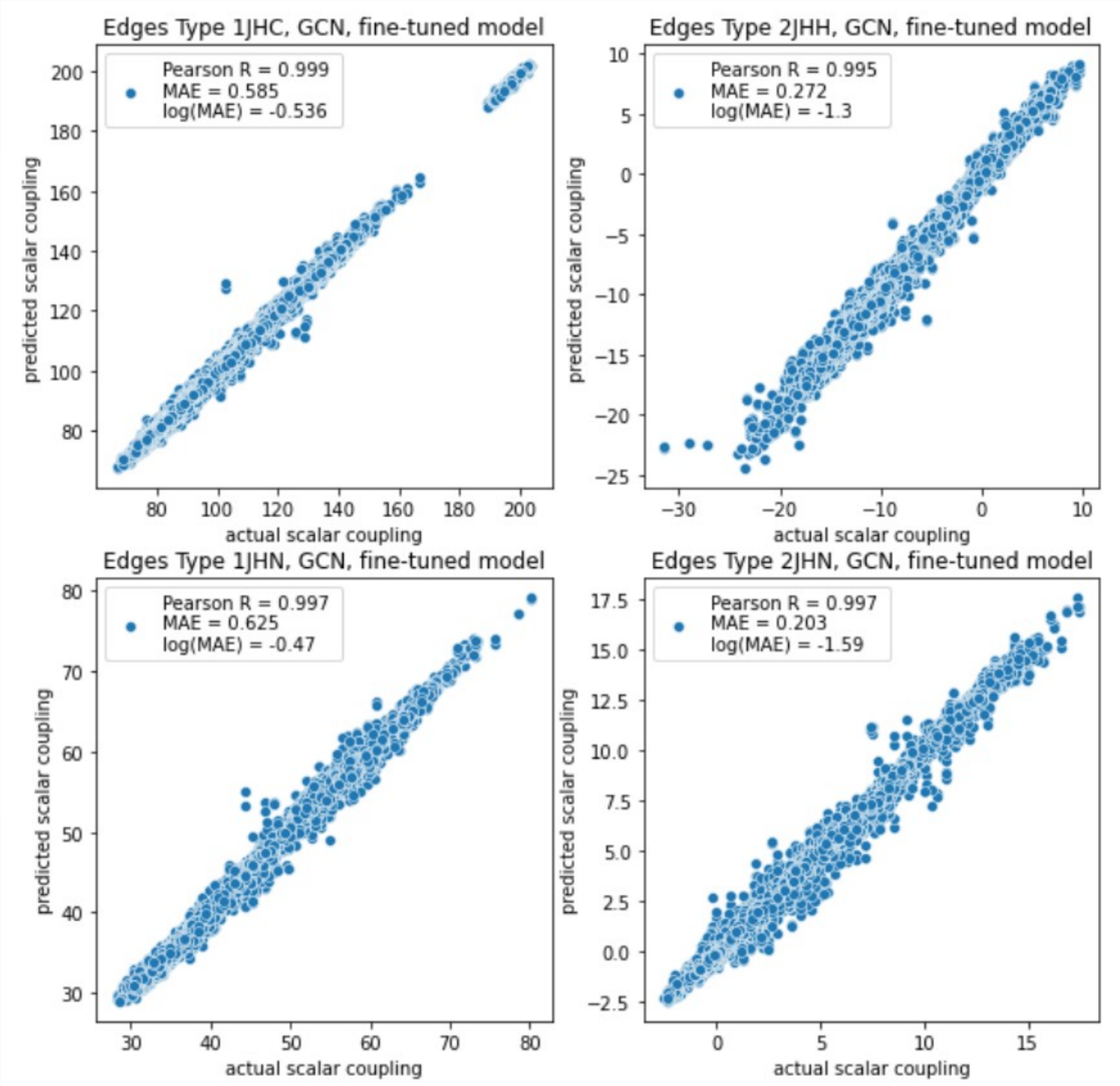
Results – Global model



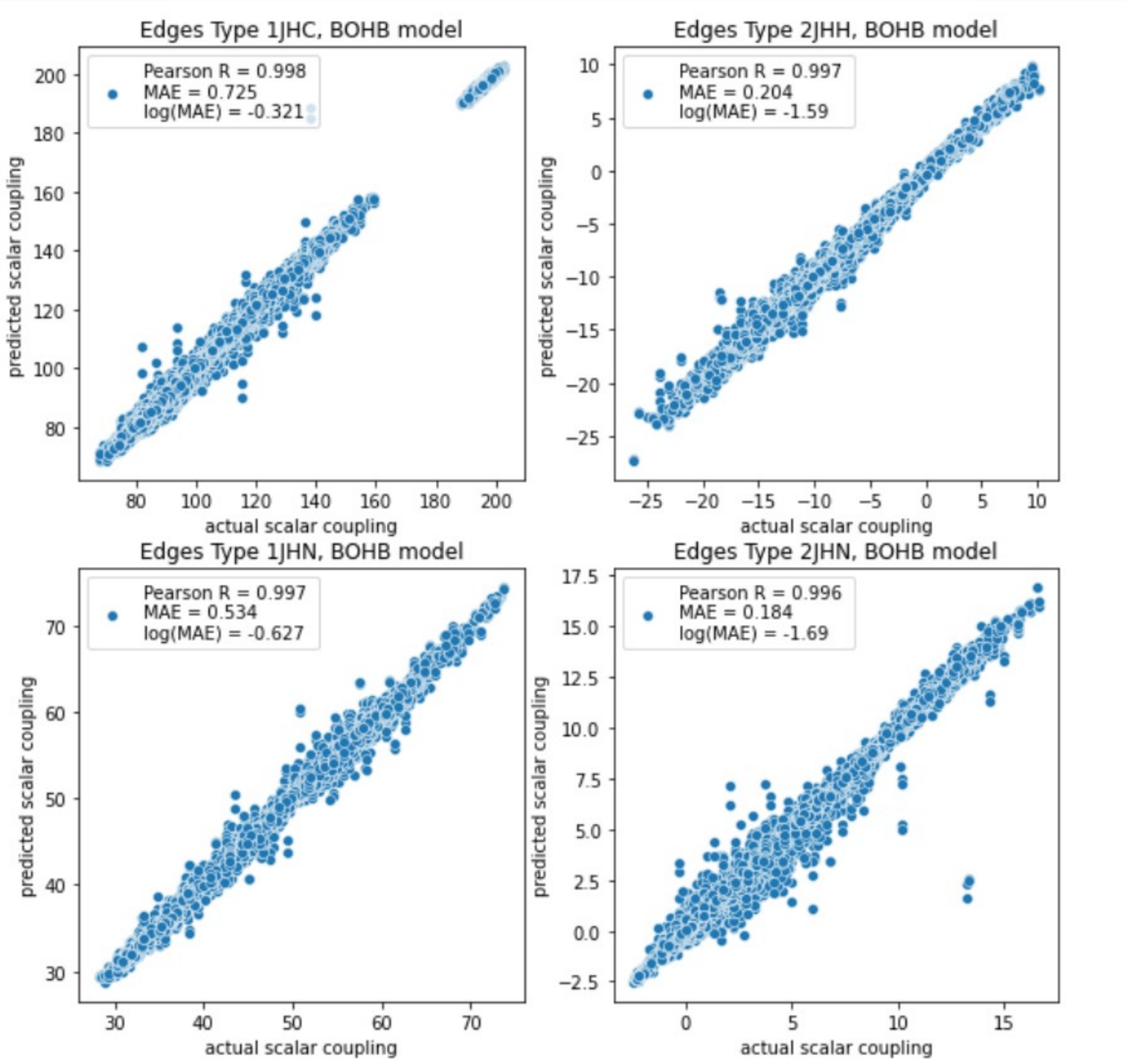
Results – Concat embeddings



Results – Fine-tuned embeddings



Results – BOHB



Results – Summary

		1JHC	2JHH	1JHN	2JHN	Average
R	Baseline	0.902	0.492	0.911	0.708	0.75325
	Global Model	0.999	0.995	0.997	0.997	0.997
	Tabularized	0.998	0.998	0.996	0.997	0.99725
	Fine-tuned	0.999	0.995	0.997	0.997	0.997
	BOHB	0.998	0.997	0.997	0.997	0.99725
MAE	Baseline	5.29	2.21	4.06	1.99	3.3875
	Global Model	0.725	0.272	0.625	0.203	0.45625
	Tabularized	0.737	0.166	0.718	0.165	0.4465
	Fine-tuned	0.585	0.272	0.625	0.203	0.42125
	BOHB	0.725	0.204	0.534	0.184	0.41175
Log(MAE)	Baseline	1.67	0.791	1.4	0.69	1.13775
	Global Model	-0.321	-1.3	-0.47	-1.59	-0.92025
	Tabularized	-0.305	-1.8	-0.331	-1.8	-1.059
	Fine-tuned	-0.47	-1.3	-0.47	-1.59	-0.9575
	BOHB	-0.321	-1.59	-0.627	-1.69	-1.057

Conclussions

- Performance higher than baseline (Pearson-R 0.99 vs 0.75).

- Performance (and useful features) differ among target types. Type 1 edges are more difficult to predict, as they have larger variance.

- Tabularization has bigger effect for type 2 edges.

- Fine-tuning has bigger effect for type 1 edges.

- Hyperparameter optimization affected all edge types.

- Performance far from Kaggle leaderboard (LOG-MAE -1.06 vs -3.23968): Proper hyperoptimization of all models + feature engineering + ensemble of best models following different strategies could fill the gap.

Thank you for your attention!