
Graph Neural Networks to Clean Empirical Covariance Matrices and Applications to Portfolio Optimization

Lorenzo Villassero

Home Supervisor: Alfredo Braunstein
Host Supervisor: Christian Bongiorno

Master's degree in Physics of Complex Systems
Academic Year: 2025-26



UNIVERSITÉ
FRANCO
ITALIENNE
UNIVERSITÀ
ITALO
FRANCESE



**Politecnico
di Torino**
International
University



université
PARIS-SACLAY



**SCIENCES
SORBONNE
UNIVERSITÉ**

**Université
Paris Cité**

Abstract

Portfolio optimization relies on accurate covariance matrix estimation; however, empirical matrices are often noisy and can lead to suboptimal asset allocations. Traditional cleaning methods are based on Random Matrix Theory and rely on the assumption of rotational invariance. While mathematically convenient, they completely neglect the underlying structure of financial markets: they focus on cleaning the eigenvalues of the covariance matrix but ignore the eigenvectors. However, heuristic methods have shown that hierarchical structures exist in financial markets and that they can be used to clean covariance matrices. This thesis explores the possibility of moving beyond the rotational invariance assumption. We propose a custom Graph Neural Network that learns the underlying hierarchy of financial markets from noisy data and uses it to produce a cleaner covariance matrix. The core idea is to represent assets as nodes and their pairwise dependencies as edges. The model employs message-passing layers to capture dependencies and uses a dynamic halting mechanism that exploits oversmoothing to capture high-level interactions. Initial tests on synthetic and real data indicate that this Graph Neural Network approach matches or surpasses the performance of state-of-the-art methods. The network is found to be expressive enough to learn the market organization at each scale of the hierarchy. Being intrinsically non-rotationally invariant, this Graph Neural Network approach serves as a proof of concept, showing that it is possible to clean the informative eigenvectors by exploiting the underlying structure of financial markets. This work opens the door to a new class of data-driven methods in which the structure emerging from the data is exploited to clean the informative eigenvectors of the covariance matrix, overcoming the limitations of rotational invariance.

Keywords: Econophysics, Graph Neural Networks, Information Theory, Portfolio Optimization

Contents

1	Introduction: Problem Statement and Definitions	1
1.1	Portfolio Optimization	1
1.2	Covariance Matrix Estimation	2
1.2.1	Rotationally Invariant Estimators and the Oracle	3
1.2.2	State of the art Covariance Estimator	3
1.3	Contribution of This Work	4
2	Graph Neural Network Architecture	6
2.1	From Covariance Matrix to Graph	6
2.2	Model Architecture	8
2.3	Input/Encoder Layer	8
2.4	Cleaning Layer	9
2.4.1	Message Passing Branch	10
2.4.2	Halting Branch	10
2.5	Decoder	11
2.6	Loss Function	11
3	Fine-tuning	12
3.1	Synthetic Datasets	12
3.2	Training and Test Procedure	13
3.3	Hyperparameters	13
3.4	Tuning Procedure	14
3.5	Tuning Results	16
4	Results	17
4.1	Results on Synthetic Datasets	17
4.1.1	Scaling With the Matrix Size	20
4.1.2	Scaling With the Length of the Time Series	22

4.2	Results on Real Dataset	23
4.3	Portfolio Optimization	25
5	Conclusions	29
5.1	A Unified Approach	29
5.2	Conclusion and Future Work	30
A	Synthetic Datasets	32
A.1	Mix Dataset	32
A.2	Overlapping Dataset	32
B	Returns and Drifted Weights	34
C	Results on Synthetic Datasets	35
D	Results on Real Datasets	36
E	Results on Long-Only Portfolios	37
F	Residual Learning	39
G	KLD Loss	44

Chapter 1

Introduction: Problem Statement and Definitions

1.1 Portfolio Optimization

Everyone with some money to invest faces the same question: where should it go? It is intuitively clear that spreading capital across different sectors offers some protection against risk. But a deeper question arises: is there an optimal way to allocate it? The goal of portfolio optimization is to find the optimal allocation of capital across a set of assets. In order to formalize this problem, we need to define some key concepts and notations. We can define the linear return of an asset i at time t as the relative change in the price of the asset from one time period to the next. Mathematically, it can be expressed as:

$$r_{i,t} = \frac{P_{i,t} - P_{i,t-1}}{P_{i,t-1}} \quad (1.1)$$

where $P_{i,t}$ is the price of the asset at time t and $P_{i,t-1}$ is the price of the asset at time $t-1$. The return can be positive or negative, depending on whether the price of the asset has increased or decreased. We have capital $\mathcal{W}$ and we want to allocate it across N assets. We can represent the allocation of capital as a vector $w = (w_1, w_2, \dots, w_n)$, where w_i is the fraction of capital allocated to asset i . The sum of all allocations must be equal to $\mathcal{W}$, i.e.:

$$\sum_{i=1}^n w_i = \mathcal{W} \quad (1.2)$$

A convenient choice is to normalize the capital $\mathcal{W}$ to 1. We can define the return and the variance of the portfolio p at time t as follows:

$$r_t = \sum_{i=1}^n w_i r_{i,t} \quad (1.3)$$

and

$$(\sigma_t)^2 = \sum_{i=1}^n \sum_{j=1}^n w_i w_j \Sigma_{ij} \quad (1.4)$$

where Σ_{ij} is the covariance between the returns of assets i and j .

In 1952, Markowitz [22] put forward the idea that risk-averse investors should optimize their portfolio based on a combination of two objectives: expected return and risk. Mathematically the expected return of the portfolio R_t measures the expected benefit, while the most basic risk measure is the variance $(\sigma_t)^2$: a higher variance means that there are fatter tails in the distribution of the returns, which may cause big losses. In Markowitz's theory construction is framed as a bi-objective optimization problem with the expected return and the risk measured by the variance optimized simultaneously. In

the literature, there exist several ways to formulate this optimization problem under different kinds of constraints. We will introduce the case in which the only constraint is weight normalization, also known as the budget constraint. A detailed analysis of other more advanced variants of this problem can be found in Ref. [7] and in Ref. [23]. Introducing a parameter δ that controls how risk-averse the investor is, the problem can be stated as follows:

$$\max_w \left(w^\dagger \mu - \frac{\delta}{2} w^\dagger \Sigma w \right) \quad (1.5)$$

$$w^\dagger \mathbf{1} = 1 \quad (1.6)$$

where μ is the vector of expected returns. This particular case is well known in the literature and using the Lagrange multipliers method we can even write the closed-form solution of the problem:

$$w^* = \frac{1}{\lambda} \left(\Sigma^{-1} \mu - \frac{\mathbf{1}^\dagger \Sigma^{-1} \mu}{\mathbf{1}^\dagger \Sigma^{-1} \mathbf{1}} \Sigma^{-1} \mathbf{1} \right) + \frac{\Sigma^{-1} \mathbf{1}}{\mathbf{1}^\dagger \Sigma^{-1} \mathbf{1}} \quad (1.7)$$

In this setting, this result can be interpreted as follows: the more δ increases, the more the investor is risk-averse and the more the optimal allocation will be focused on minimizing the variance of the portfolio (the second term on the right-hand side will dominate). If we send δ to infinity, we get the so-called Global Minimum Variance (GMV) portfolio, which is the portfolio that minimizes the variance without any consideration of the expected return.

$$w_{GMV} = \frac{\Sigma^{-1} \mathbf{1}}{\mathbf{1}^\dagger \Sigma^{-1} \mathbf{1}} \quad (1.8)$$

This result is particularly informative for our purposes, because it explicitly shows that the knowledge of the covariance matrix Σ is crucial. Indeed, GMV portfolio will be the primary focus of our study for two reasons. First, accurately estimating expected returns is notoriously difficult: relying on historical returns is naive since past returns are essentially uncorrelated with future ones. Second, because our ultimate objective is covariance cleaning, the GMV framework provides an ideal testing ground. Since Σ is the only input parameter that needs to be estimated, we can directly evaluate the effectiveness of our filtering techniques without the confounding effects introduced by return estimation errors.

In practice, we do not have access to the true covariance matrix, but we need to estimate it from the data. This is a crucial point, because the estimation of the covariance matrix can be very noisy. If we call $R \in \mathbb{R}^{N \times T}$ the matrix of the returns of the N assets over T time periods, we can estimate the covariance matrix as follows:

$$\hat{\Sigma} = \frac{1}{T} R R^\dagger \quad (1.9)$$

The problem is that when $T \rightarrow N$ the estimator $\hat{\Sigma}$ becomes very noisy and moreover if $T < N$, the estimator is not even invertible. This clearly will lead to a suboptimal allocation or even to an unfeasible solution. Notice that this isn't a problem that only appears in this specific formulation of the Markowitz optimization problem, but is more general. Indeed, in any case we will need to give as input the covariance matrix, and if we have a bad estimation of it, we will get a bad solution. In the following chapters we will refer to the estimated quantities with a hat, to distinguish them from the true quantities. Throughout the following, we assume returns are centered, i.e., their sample mean has been removed.

1.2 Covariance Matrix Estimation

In the last twenty years, there has been considerable research on covariance matrix estimation and cleaning [9]. When estimated from finite samples, empirical covariance matrices are ill-conditioned and affected by large estimation errors [15, 7], which leads to suboptimal portfolio allocations. Before presenting the most effective solutions proposed to address this problem, it is useful to fix the criterion by which an estimator is judged.

1.2.1 Rotationally Invariant Estimators and the Oracle

A large part of the literature restricts the attention to the class of Rotationally Invariant Estimators (RIE) [10]. The underlying hypothesis, known as the Rotational Invariance Assumption (RIA), is that in absence of any prior information on the structure of the true matrix, there is no reason to modify the eigenvectors of the sample matrix. Formally, given the eigen-decomposition of the sample correlation matrix

$$\hat{C} = \hat{V} \hat{\Lambda} \hat{V}^T, \quad (1.10)$$

any RIE takes the form

$$\hat{C}_{RIE} = \hat{V} \Lambda^* \hat{V}^T, \quad \Lambda^* = \text{diag}(f_1(\hat{\lambda}_1 \dots \hat{\lambda}_N), \dots, f_N(\hat{\lambda}_1 \dots \hat{\lambda}_N)), \quad (1.11)$$

for some cleaning function f acting on the sample eigenvalues. To compare estimators, the standard metric is the Mean Squared Error (MSE) with respect to the true matrix $C = U \Lambda U^\dagger$:

$$\text{MSE}(\hat{C}_{RIE}, C) = \frac{1}{N} \|C - \hat{C}_{RIE}\|_F^2. \quad (1.12)$$

This is exactly the (normalized, squared) Frobenius distance. Since the Frobenius norm is invariant under orthogonal transformations, we can rotate into the basis of the estimator's eigenvectors and write:

$$\text{MSE}(\hat{C}_{RIE}, C) = \frac{1}{N} \|\hat{V}^T C \hat{V} - \Lambda^*\|_F^2 = \underbrace{\frac{1}{N} \sum_{i=1}^N (\xi_i - \lambda_i^*)^2}_{\text{MSE}_{\text{eig}}} + \underbrace{\frac{1}{N} \sum_{i \neq j} (\hat{V}^T C \hat{V})_{ij}^2}_{\text{MSE}_{\text{vec}}}, \quad (1.13)$$

where $\xi_i = \hat{v}_i^T C \hat{v}_i$ are the so-called *Oracle eigenvalues* [10]. The two terms have a clear interpretation:

- The first term is the error *removable by spectrum cleaning alone*: it measures how far the predictor's eigenvalues λ_i sit from the best values ξ_i attainable on its own axes. We will refer to it as MSE_{eig}
- The second term is the *irreducible floor* set by the eigenvectors: the off-diagonal mass vanishes iff $\hat{V}$ diagonalizes C , i.e. iff the estimator's eigenvectors coincide with the target's. No re-scaling of eigenvalues can ever reduce it. We will refer to it as MSE_{vec} .

Within the RIE class the eigenvectors are fixed to the sample ones, so MSE_{vec} is a constant and the best one can do is to set $\lambda_i^* \rightarrow \xi_i$. The resulting estimator,

$$\hat{O} = \hat{V} \Xi \hat{V}^T, \quad \Xi = \text{diag}(\hat{V}^T C \hat{V}), \quad (1.14)$$

is known as the *Oracle estimator*. It sets $\text{MSE}_{\text{eig}} = 0$. In this sense it is the best among all estimators sharing the sample eigenvectors. It is obvious that we do not have access to the Oracle estimator. Ξ depends directly on C , the true correlation matrix, which is unknown. Its role is therefore that of a theoretical benchmark: the lowest MSE any RIE can possibly achieve. Even if the exact knowledge of the Oracle is not possible, a series of results shows that under certain constraints we can asymptotically approach it [10, 16]. In the high-dimensional limit $N, T \rightarrow \infty$ with their ratio fixed, and under suitable conditions on the moments of the returns, the expectation of the Oracle converges to a deterministic function of the sample eigenvalues. This is at the theoretical foundation of the non-linear shrinkage estimators reviewed below.

1.2.2 State of the art Covariance Estimator

Once we have introduced these general results, we can discuss the different approaches to covariance estimation:

RMT-based approaches. These methods rely on Random Matrix Theory to separate true economic signals from statistical noise. They belong to the RIE class presented above. One of the earliest techniques is eigenvalue clipping: Ref. [15] showed that the bulk of the spectrum of empirical financial correlation matrices closely follows the Marchenko–Pastur distribution expected for pure noise. Eigenvalues falling below the upper edge of this distribution are therefore treated as uninformative and replaced by a constant average value, while those above the threshold are retained [7]. A second approach worth mentioning is the linear shrinkage estimator of Ref. [17], which takes a convex combination of the sample matrix and a diagonal target; it remains well-defined even when $N > T$, but its cleaning function is linear in the sample eigenvalues, which is optimal only under restrictive assumptions. The modern estimators are grounded on the asymptotic Oracle discussed above [18]. There exist several implementations, we will use the recent QIS [19], which is currently considered the state of the art in financial covariance filtering. Despite their solid theoretical foundations, these methods have structural limitations. By construction they trust the sample eigenvectors, which in practice are contaminated by noise. In this sense they completely ignore the structural information contained in the data. As underlined before there is no eigenvalue correction that can lower the MSE on the eigenvectors. Moreover, the optimality guarantees hold in the thermodynamic limit and under moment conditions on the returns that are questionable.

Hierarchical approaches. These methods work on the assumption that the market is organized in a hierarchical structure [21], with assets naturally grouping into sectors and sub-sectors, an assumption supported by strong empirical evidence. By applying topological filtering to remove weak links, they extract a robust “skeleton” used to reconstruct a cleaned correlation matrix [27]. Unlike RIEs, hierarchical filters modify both the eigenvalues and the eigenvectors of the sample matrix. A state-of-the-art representative of this family is the Bootstrapped Average Hierarchical Clustering (BAHC) estimator [4, 2]. This estimator averages hierarchically filtered correlation matrices over bootstrap replicas of the data, thus reducing the sensitivity to the specific dendrogram extracted from a single noisy sample. The price to pay is the rigidity of the underlying ansatz. Indeed a tree-like structure may be too restrictive to capture the complex, overlapping interdependencies that characterize modern financial markets.

Machine learning approaches. More recently a new promising line of research has been opened. The novelty is that the cleaning function is not derived analytically but learned from data. A notable example is the rotation-invariant neural network called RIEnet [6]. It parameterizes the eigenvalue cleaning function $\mathbf{f}$ of Eq. 1.11 with a bidirectional LSTM acting on the ordered sample spectrum, so that the non-separability and permutation equivariance of the target map are respected by construction. The network is trained end-to-end on realized out-of-sample portfolio variance rather than on the Frobenius loss. By construction, however, it still belongs to the RIE class, since it retains the sample eigenvectors.

1.3 Contribution of This Work

The gap addressed in this thesis is the following: we propose a data-driven estimator that acts on the full correlation matrix, modifying eigenvalues and eigenvectors jointly, without relying neither on the RIA nor on a rigid hierarchical ansatz. We take inspiration from recent advances in machine learning applied to finance [25, 13]. We employ a Graph Neural Network (GNN) [29] to extract structural information from the data. Since such architectures cannot be applied directly to a correlation matrix, our first task was to devise a suitable way of encoding the information contained in the correlation matrix into a graph. Once this was done, the network proved expressive enough to capture the complex relationships among assets across all scales [30]. Clearly, in doing so, we had to address well-known issues in the GNN literature, most notably oversmoothing [20]. This phenomenon arises when several layers are applied repeatedly to the same graph, causing node representations to converge toward a

uniform distribution in which no structure can be distinguished. We decided to handle this problem dynamically through a halting mechanism, which allows the network to generalize well without fixing a predetermined number of message-passing steps. Finally, we designed a decoding procedure capable of mapping the output of the network back into a valid correlation matrix.

We will discuss both strengths and weaknesses of our architecture. We believe this approach can serve as a valuable tool for moving beyond rotational invariance: within a broader framework, it can be used as an investigative instrument to understand which structures, across all scales, remain constant and stable over time, and which instead evolve. This observation is crucial, as it opens the door to a new class of estimators that incorporate corrections to the eigenvectors as well. For this reason, we refer to them as Rotationally Sensitive Estimators (RSE). In the following we will first introduce the architecture of our model in Chapter 2. Part of the work done to fine-tune the model will be synthesized in Chapter 3. Then we will present some results on synthetic data and some preliminary results on real data in Chapter 4. In the last Chapter 5 we will discuss the limitations of our approach and possible future developments.

Chapter 2

Graph Neural Network Architecture

To tackle the problem of cleaning the correlation matrix, the idea is to use a GNN. Clearly, to use a GNN, we need a graph. The first section of this chapter will be devoted to understanding how we can transform a covariance matrix into a graph that the GNN can actually process. The remainder of the chapter will present the model and its technical aspects.

2.1 From Covariance Matrix to Graph

Covariance matrices are a useful tool for representing relationships between assets. However, because their values are scale-dependent, they indicate the direction of a relationship, but make it difficult to assess its true strength. To overcome this problem, we can use the normalized version of it, i.e. the correlation matrix, which is defined as follows:

$$C_{ij} = \frac{\Sigma_{ij}}{\sqrt{\Sigma_{ii}\Sigma_{jj}}} \quad (2.1)$$

Once we have the correlation matrix, we can use it to build a graph. Correlation matrices can be directly converted into distance matrices as in Ref. [27] using the following formula:

$$D_{ij} = \sqrt{2(1 - C_{ij})} \quad (2.2)$$

This is a particularly advantageous choice, as it guarantees that D is a Euclidean distance metric. We want nodes that are close in the correlation-induced distance to interact strongly, and nodes that are far apart to be effectively decoupled. The adjacency matrix should therefore represent local interactions whose strength decays with distance. We define a weighted adjacency matrix to encode the weight of the connections between node i and node j as:

$$K_{ij} = \exp(-D_{ij}^2/\tau) \quad (2.3)$$

where τ is a hyperparameter that controls the scale of the distances. In Figure 2.1 we can see how the distance changes as a function of the correlation, while in Figure 2.2 we can see how the exponential kernel maps the distance to the weight of the edges. From Figure 2.2 it is clear that the exponential kernel assigns a weight close to one to pairs of assets separated by a small distance, and suppresses it smoothly as the distance grows, so that only sufficiently similar assets remain meaningfully connected. Moreover, in order to make the graph more sparse while keeping the whole process differentiable, we can use a mask function, which is defined as follows:

$$M_{ij} = \frac{1}{1 + \exp(\frac{(D_{ij} - \text{cutoff})}{\sigma})} \quad (2.4)$$

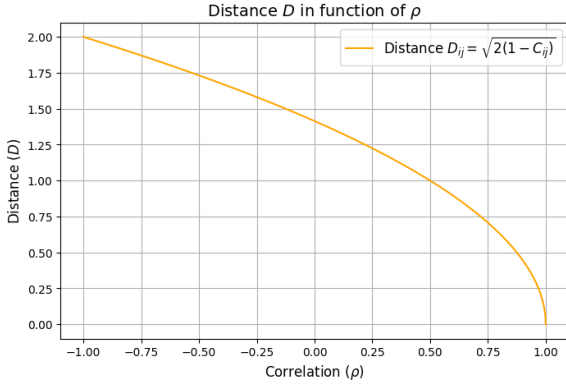


Figure 2.1: Plot of the mapping from correlation to distance.

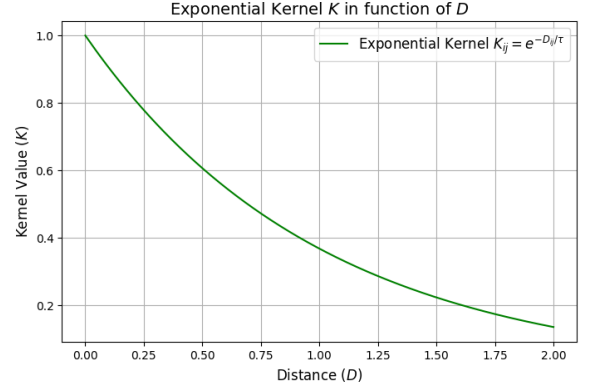


Figure 2.2: Plot of the exponential mapping from distance to weight.

where cutoff is a hyperparameter that controls the sparsity of the graph, while σ is a hyperparameter that controls the smoothness of the mask. The mask function complements the kernel by controlling how and where the interaction is switched off. The final weighted adjacency matrix of the graph is then given by:

$$A_{ij} = M_{ij} \cdot K_{ij} \quad (2.5)$$

In Figure 2.3 we can see what the final mapping from correlation to weight of the edges looks like,

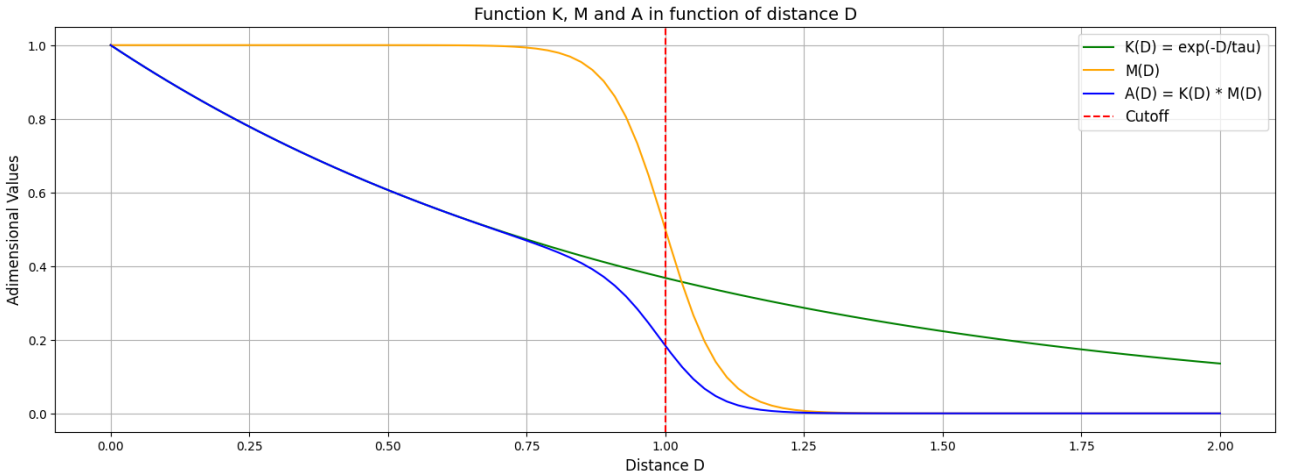


Figure 2.3: Graphical representation of the behaviour of the functions $A(D)$, $M(D)$ and $K(D)$ as a function of the distance D . The parameters are set as follows: $\tau = 1$, cutoff = 1.0 and $\sigma = 0.05$.

where we can see how the mask function makes the graph more sparse.

This graph representation of the covariance matrix is what allows us to use graph neural networks to learn the structure of the graph and to perform the cleaning of the covariance matrix. Moreover, this construction has a natural interpretation in terms of the structure. A cluster of assets corresponds to a group of stocks whose mutual distances are, on average, smaller than their distances to the rest of the market. Eq. 2.5 enforces this condition, ensuring that these stocks end up strongly connected to one another and weakly connected to the outside. The adjacency matrix therefore imposes no cluster assignment a priori: it defines a set of locally interacting objects, and the collective, group-level behaviour emerges bottom-up from these pairwise interactions. Notice that the parameters τ , cutoff and σ are all learnable, which means that the model can learn to adapt the graph structure to the data. It is worth stressing that the central idea of this work is already visible at this stage: collective structure is built up from local interactions, not assumed.

2.2 Model Architecture

The model we propose is based on three main components: the input/encoder layer, the cleaning layer and the decoder. The general working principle of the model is as follows. We feed the sample correlation matrix into the input layer, which prepares the feature vectors for each node together with any attributes passed as input. The cleaning layer is then responsible for the actual cleaning of the graph, while the decoder takes the cleaned features and reconstructs the cleaned correlation matrix. In

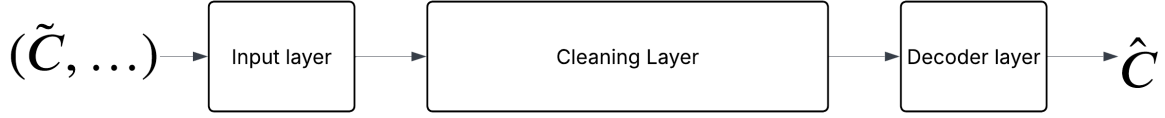


Figure 2.4: High level architecture of the proposed model.

Figure 2.4 we can see a high-level diagrammatic representation of our model. In the following sections a detailed analysis of each component will be provided, starting from the input layer, then the cleaning layer and finally the decoder.

2.3 Input/Encoder Layer

The primary objective of the input layer is to transform the raw empirical data into a structured representation. Specifically, it prepares the necessary structural embeddings for the subsequent processing steps. The layer takes as its mandatory input the sample correlation matrix computed from the in-sample data. Furthermore, the architecture is designed to flexibly incorporate supplementary information, which in the context of our model we call attributes. There are three types of attributes:

- Global attributes, such as the ratio $q = \frac{N}{T}$, where T is the number of observations and N is the number of assets, which we know is useful information to understand how noisy the covariance matrix is [8].
- Node attributes, such as the sector they belong to, their market capitalization, etc.
- Edge attributes, such as the sign of the correlation, mutual information, etc.

In Figure 2.5 we show a diagram of the input layer, illustrating the main components as well as its inputs and outputs. The input layer is composed of three independent blocks. First, the Distance conversion block converts the correlation matrix into a distance matrix using relation (2.2). Second, the Features encoder initializes a feature vector for each node. Finally, the Attributes encoder encodes all the attributes into a latent space.

Following an approach similar to attention mechanisms in Ref. [28], each node's feature vector is initialized as an independent vector in the Fourier space. For node i , the normalized index is $\tilde{i} = \frac{i}{\max(N-1, 1)}$ and the angles are $\theta_{i,k} = 2\pi \cdot \tilde{i} \cdot f_k$. The frequencies f_k scale logarithmically up to $\max(\frac{N}{2}, 1)$. The initial feature in matrix X_0 is:

$$X_{0,i} = \left[\tilde{i}, \sin(\theta_{i,0}), \cos(\theta_{i,0}), \dots \right]^\top \quad (2.6)$$

truncated to dimension F . This choice is arbitrary, and other initialization strategies that provide a unique representation for each node could be used as well. In the end the outputs of the input layer are the distance matrix D , the embedded attributes E and the initial feature vectors for each node X_0 . An important point to emphasize is that both the dimension of the feature vectors F and the dimension of the embedded attributes are hyperparameters that can be tuned. Up to this point, we have not yet built the graph; we have only prepared the components needed for the next layer.

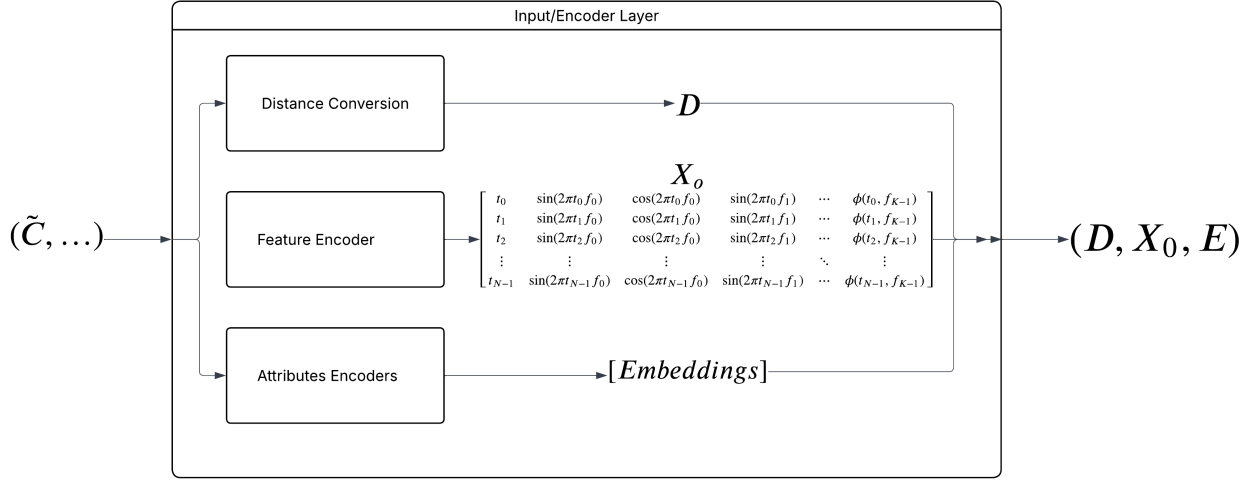


Figure 2.5: Schematic representation of the input layer

2.4 Cleaning Layer

The cleaning layer is the core of the model, it is responsible for updating the feature vectors of the nodes by recursively aggregating information from their neighbours. Ultimately, its main purpose is to filter out the noise and allow the underlying structure to emerge at each scale. In Figure 2.6 a detailed

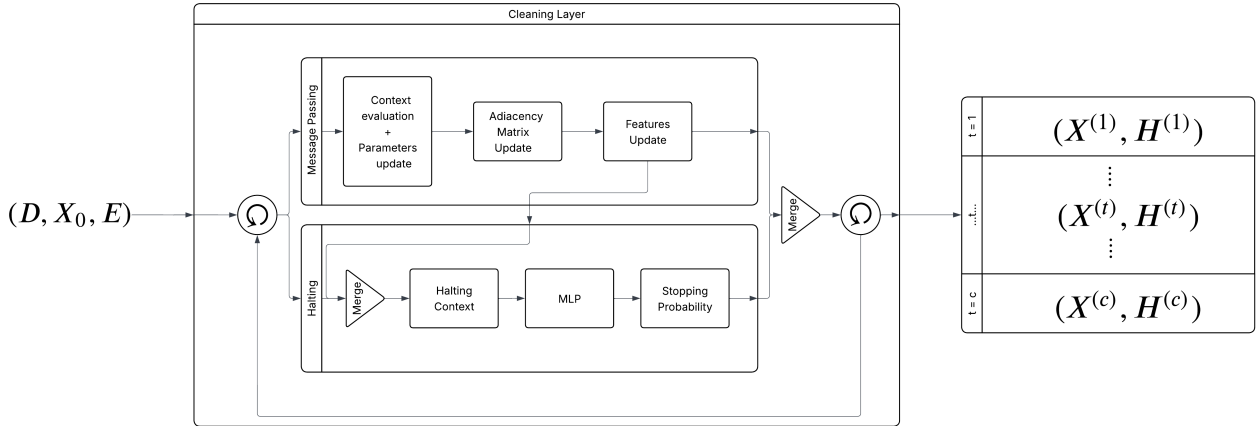


Figure 2.6: Schematic representation of the cleaning layer. It takes as input the output of the input layer and it returns for each layer of the GNN a sequence of feature vectors $X^{(t)}$ and the halting probability of the layer $H^{(t)}$

diagram of the cleaning layer is provided. We can see that there are two branches:

- **Message passing branch:** it updates the feature vector of each node by propagating and aggregating information along the edges of the graph. This is where the actual cleaning takes place, one refinement step at a time.
- **Halting branch:** it determines how many iterations are actually needed to extract all the relevant information from the graph. Rather than fixing the number of steps in advance, the model learns the cleaning operation step by step, deciding on its own when to stop. A key advantage of this design is flexibility at inference time: when facing a particularly complex or large matrix, the maximum number of steps can be increased without any retraining.

In the following we will use the $(\cdot^{(t)})$ notation to indicate the iteration step. We begin by analyzing the first branch, which is the message passing branch.

2.4.1 Message Passing Branch

At each iteration step (t) , we have the input $(D^{(t)}, X^{(t)}, E^{(t)})$. The first thing to do is to evaluate a global context vector g which encodes our current understanding of the network. From $D^{(t)}$, $X^{(t)}$ and $E^{(t)}$ we extract summary statistics (e.g. mean and extreme values of the distances, features and attributes) and concatenate them into a global context vector g . This vector g is then used as input for a Multi-Layer Perceptron (MLP) which will output the correction we need to apply to the parameters of the graph, i.e. τ , cutoff and σ . Denoting a generic parameter by θ , and the corresponding MLP output by δ_θ we have:

$$\theta^{(t)} = \theta^{(t-1)} \exp(\delta_\theta) \quad (2.7)$$

This way we can adapt the graph structure to the data, making it more sparse or more dense depending on the current understanding of the network. Once we have the updated parameters, we can build the adjacency matrix at the current time step $A^{(t)}$ using the formula (2.5). In order to update the feature vectors $X^{(t)}$ we use the well-known spatial-based GNN convolutional paradigm illustrated in Ref. [29]. This is based on propagating the information of each node to its neighbors through the edges of the graph, and then aggregating this information to update the feature vector of each node. The update rule is given by the following formula:

$$X^{(t+1)} = \phi(A^{(t)} X^{(t)} W_{\text{msg}}^{(t)} + X^{(t)} W_{\text{self}}^{(t)} + b^{(t)}) \quad (2.8)$$

where $X^{(t)} \in \mathbb{R}^{N \times F}$ are the node features at layer (time) t and $A^{(t)}$ is the adjacency matrix at time step t . The learnable parameters of the update are the message and self weight matrices $W_{\text{msg}}^{(t)}, W_{\text{self}}^{(t)} \in \mathbb{R}^{F \times F}$ and the bias term $b^{(t)} \in \mathbb{R}^F$, while $\phi(\cdot)$ is a non-linear activation function. We adopt the Swish function [24] defined as:

$$\phi(x) = x \cdot \sigma(x) = \frac{x}{1 + e^{-x}} \quad (2.9)$$

Unlike ReLU, it is smooth and non-monotonic, which is consistent with our fully differentiable design and improves gradient flow through the deep message-passing iterations. Before using these new features to recompute the global context vector g and repeat the process (up to c times, where c is a hyperparameter controlling the maximum number of iterations), we pass this information to the second branch of the layer, which is responsible for determining how many iterations are still needed.

2.4.2 Halting Branch

The halting branch quantifies how much information we have already extracted from the graph and how much more we still need to extract. It takes as input the new node features and computes a new global context vector g' , which is then passed to an MLP that outputs a scalar value representing the probability of stopping at the current time step, given the information that survived up to it. We then compute the halting probability as:

$$p_{\text{halting}} = p_{s(t)} \cdot p(t \mid s(t)), \quad (2.10)$$

where $p_{s(t)}$ is the probability of surviving up to time step t , and $p(t \mid s(t))$ is the probability of stopping at time step t given that we have survived up to it. The survival probability is then updated for the next step as:

$$p_{s(t+1)} = p_{s(t)} \cdot (1 - p(t \mid s(t))). \quad (2.11)$$

In this way we obtain a differentiable halting mechanism that allows the model to learn how many iterations it needs to perform in order to extract all the necessary information from the graph. After this, we store the new features together with their halting probability, and repeat the process until we

reach the maximum number of iterations. As we will see, this halting mechanism is crucial to prevent oversmoothing, a common problem in graph neural networks: after a certain number of iterations, the node features become too similar to one another, and we lose the ability to distinguish between them.

2.5 Decoder

The decoder is responsible for interpreting the output of the cleaning layer, mapping the graph representation back into a cleaned correlation matrix at each time step. We start by precomputing a common basis in the Fourier space of dimension k , which is a hyperparameter that can be tuned. This k -dimensional basis is then used to project the features of each node at each time step. At each time step t , the nodes features vectors are given to a dense layer that performs this projection. Then this projection is L^2 -normalized to a unit vector, so it encodes only the direction. A second head outputs a per-node scalar $e_i \in (0, 1)$, and the unit vector is scaled by $\sqrt{e_i}$ so that the squared norm of row i equals e_i . This scalar, therefore acts as a learned per-node weight: it lets the model down-weight nodes whose correlation pattern it is less confident about. Stacking over the N nodes gives a matrix $Z^{(t)} \in \mathbb{R}^{N \times k}$. At each time step we can then compute a cleaned correlation matrix as the outer product of $Z^{(t)}$ with itself:

$$\hat{C}^{(t)} = Z^{(t)} Z^{(t)T} \quad (2.12)$$

whose diagonal we then reset to one. Before this reset, the off-diagonal entries are $\hat{C}_{ij}^{(t)} = \sqrt{e_i e_j} \cos \theta_{ij}$, so the row energies act as a per-node shrinkage on the reconstructed correlations. By construction $\hat{C}^{(t)}$ is positive semi-definite, with off-diagonal entries in $[-1, 1]$ and unit diagonal. In the end the halting probabilities are used to weigh the contribution of each time step to the final cleaned correlation matrix, which is given by the following formula:

$$\hat{C} = \sum_{t=1}^c p_{\text{halting}}^{(t)} Z^{(t)} Z^{(t)T} \quad (2.13)$$

Because this is a convex combination of positive semi-definite matrices with unit diagonal, $\hat{C}$ is itself a valid correlation matrix by construction: symmetric, positive semi-definite, with unit diagonal and off-diagonals bounded in $[-1, 1]$. The halting probabilities as seen before tell us how much information we have extracted at each time step, and therefore how much we should weigh the contribution of each time step to the final cleaned correlation matrix. At the end the decoder outputs the cleaned correlation matrix $\hat{C}$, which is then used to compute the loss function and to update the parameters of the model.

2.6 Loss Function

Our proposed model is trained end-to-end, i.e. all its components are optimized jointly through a single loss function computed on the final output. The loss we minimize is the following:

$$\mathcal{L} = \frac{1}{N^2} \sum_{i=1}^N \sum_{j=1}^N (C_{ij} - \hat{C}_{ij})^2 + \frac{\gamma}{c} \sum_{t=1}^c p_{\text{halting}}^{(t)} t \quad (2.14)$$

The first term is the mean squared error (rescaled Frobenius norm) between the out of sample correlation matrix C and the cleaned correlation matrix $\hat{C}$, while the second term is a regularization term that penalizes the model for using too many iterations, where γ is a hyperparameter that controls the strength of this regularization. Intuitively, the second term keeps the expected number of iterations low. We want to exploit a controlled amount of oversmoothing to capture high-level interactions between nodes, while avoiding the information loss caused by too many iterations. As a result, only a few iterations carry significant halting probability; the remaining ones stay small and contribute just the additional smoothing that is needed.

Chapter 3

Fine-tuning

Before turning to the results produced by our architecture, it is necessary to provide a clear description of the experimental setup. In Section 3.1, we describe the datasets used. Subsequently, the specific protocols for training and testing are clearly defined in Section 3.2. Finally, Section 3.3 presents our hyperparameters tuning strategy and the results of this procedure.

3.1 Synthetic Datasets

To train and evaluate our architecture, we construct¹ a synthetic dataset designed to mirror some of the real-world challenge of estimating correlation matrices from noisy data. These challenges are the noise arising from the finite length of the time series and the presence of a hierarchically nested correlation structure among assets. We recall that, even when such a structure is present, RIE cannot exploit it. By construction, they modify only the eigenvalues of the empirical matrix while retaining its eigenvectors, and are therefore blind to any structural information encoded in the eigenvector organization. The primary objective is thus to teach the model to recover the underlying, true market topology from a noise-corrupted input.

In the base setting, the dataset consists of input–target pairs in which the target is a “clean” correlation matrix and the input is a noisy version of it estimated from a finite sample. We start by sampling a population correlation matrix C from a nested hierarchical model, i.e. a tree-like structure; this C represents the ground truth that the model must learn to reconstruct. For each sample we then choose a number of observations T (either fixed or drawn from a range) and generate a noisy realization by simulating T return vectors drawn from a distribution with that correlation, $r_t \sim \mathcal{N}(0, C)$. From these we build the empirical covariance estimate

$$\hat{\Sigma} = \frac{1}{T} R R^\top, \quad R \in \mathbb{R}^{N \times T}, \quad (3.1)$$

which is then normalized into the corresponding correlation matrix $\hat{C}$. The effect is that when T is of the order of N (or smaller), $\hat{C}$ is heavily distorted by sampling noise, whereas for $T \gg N$ it tends to C . Each element of the dataset is therefore the pair $(\hat{C}, C)$. By varying N , T and the hierarchical structure, we obtain a heterogeneous set of examples covering different noise levels and different market organizations. We used two other categories of synthetic datasets besides the one described here. More details can be found in the Appendix A. The first one is a dataset in which the target matrix is a convex blend of the clean hierarchical structure and an unstructured inverse-Wishart correlation matrix. The second one instead is a dataset in which each target matrix is generated as a combination of two different hierarchical structures, with a random convex blend. In the following

¹The synthetic datasets were generated using libraries developed within the research group; the author’s contribution in this regard was minimal. The author adapted them for use in this specific case and modified the algorithms to produce the desired test sets

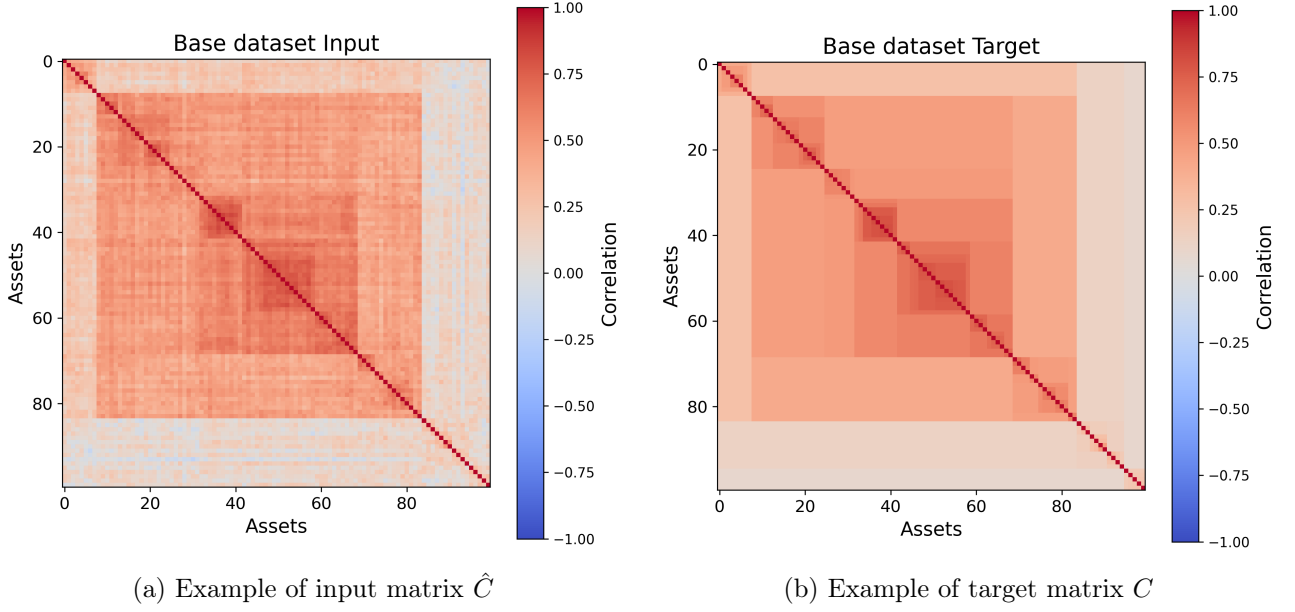


Figure 3.1: Example of a pair of sample and target correlation matrices from the base dataset.

we will refer to these three datasets as *base*, *mix* and *overlap* respectively. If not specified otherwise, the results presented in the next sections refer to the base dataset. Moreover, the default number of assets is set to $N = 100$ and the number of observations is chosen to be $T = 150$, which corresponds to a value of $q = N/T = 0.66$.

3.2 Training and Test Procedure

We train the model using the Adam optimizer [14] with a learning rate η . We set the maximum number of epochs to 300 and for each epoch we perform 100 training iterations. In each iteration we sample a new batch of input-target pairs from the training set. The size of the batch is set to 70, which is a good compromise between computational efficiency and stability of the training process. We employ an early stopping mechanism based on the validation loss, with a patience of 20 epochs, and we restore the weights of the best epoch. The training and validation sets are generated separately, so that the model is tested on unseen examples during training. To assess the performance of a trained model, we generate a test dataset of 500 examples and evaluate the metrics we need on it. Moreover to account for the stochasticity of the training process, for each model configuration we train four different instances with different starting seeds. Then for each instance we first average the metrics over the test set and then we average the results across the four instances. Unless otherwise stated, we report the mean across seeds together with its standard deviation.

3.3 Hyperparameters

In Chapter 2, we introduced the hyperparameters of our model. The performance of the model significantly depends on their values, and tuning them appropriately is essential for a meaningful application. In order, we have:

- c : the maximum number of iterations of the GNN.
- γ : the intensity of the penalty term.
- F : the size of the feature vector used to represent the input data, which can impact the model's ability to capture relevant information.

- k : the size of the common basis used in the model.
- η : the learning rate, i.e. the step size used in the optimization process.

We will now discuss the tuning process for these hyperparameters and present the results obtained on synthetic datasets.

3.4 Tuning Procedure

This section details the strategy employed to select the optimal configuration for our model. The model depends on five hyperparameters, so a grid search is not feasible. We adopted a sequential one-at-a-time (OAT) tuning procedure [11]. This means that we tune a single hyperparameter at a time, keeping the others fixed to reasonable values. This amounts to a first-order approximation of the optimization landscape, in which the individual effect of each hyperparameter is captured while interaction terms are neglected. Specifically, we proceeded as follows:

- We first tuned the maximum number of iterations c and the learning rate η since they control the training dynamics and stability.
- We then tuned the representational capacity of the model by keeping the penalty term (γ) fixed and tuning the size of the feature vector F and the size of the common basis k separately, each around the baseline while the other is held fixed.
- Finally, we tuned the penalty term γ to balance the fit to the data and the regularization.

In the following we will present the results of this tuning process. We are aware that this procedure is greedy and not optimal [26], but it is computationally efficient and provides a good starting point for further fine-tuning. We start with $c = 10$ and $\eta = 0.001$, on the basis of a small set of preliminary exploratory runs. To fix the dimension of the feature vector F we fixed the penalty term and the size of the common basis k to a baseline value ($\gamma = 0.001, k = 100$) and we tested $F \in \{16, 32, 64, 128, 256\}$. In Figure 3.2 we can see the performance of the model in terms of Mean Squared Error (MSE) between the target and the predicted matrix. The performance improves as F increases. We set $F = 256$ since it provides the best performance in terms of MSE and is not computationally prohibitive. It is

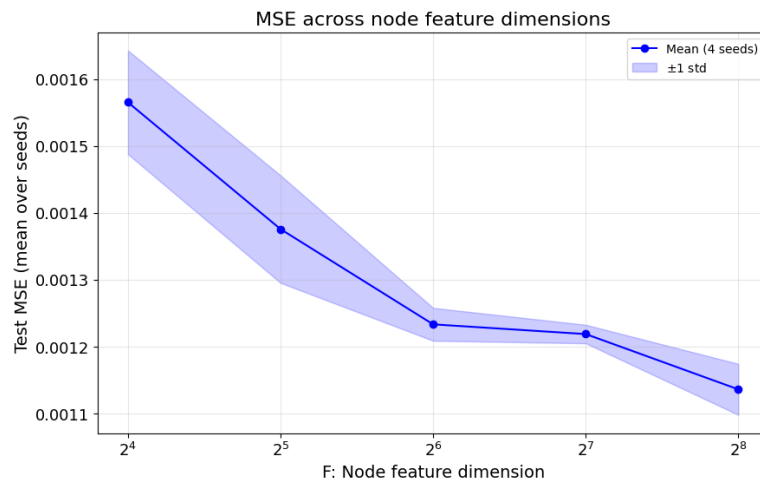


Figure 3.2: Mean Squared Error on the test set for different values of the feature vector size F . The error bars represent the standard deviation across seeds.

important to underline that similar conclusions can be drawn by looking at the performance in terms of MSE on the other datasets. The next parameter we need to fix is k . From Eq. 2.12 it is clear that

k limits from above the number of factors the model is allowed to use to represent the correlation structure. We would like not to have to set this parameter. Ideally, k should be equal to the number of assets N and the network should Figure out on its own how many factors are actually needed. To verify this we used the base dataset setting the parameters in such a way that, on average, exactly eight factors are needed to describe the target correlation matrices. We then trained the model for $k \in \{8, 16, 32, 64, 100\}$, keeping all the other hyperparameters fixed. The MSE on the test set is shown

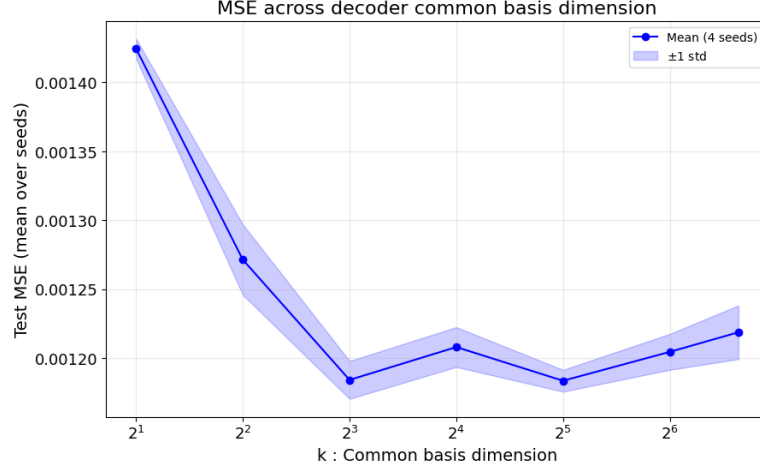
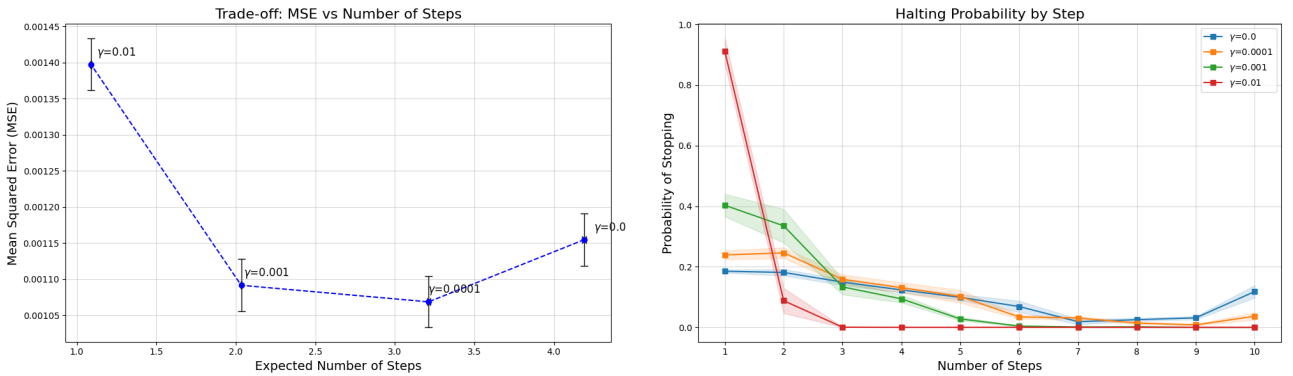


Figure 3.3: Mean Squared Error on the test set for different values of the common basis size k . The error bars represent the standard deviation across seeds.

in Figure 3.3. We can clearly see that it reaches its minimum at $k = 8$, which is the true number of factors in the data, and then it flattens out for higher values of k . We now understand precisely what k controls, and it can be exploited whenever we have prior knowledge about the effective number of factors in the data. Moreover, since an oversized basis is not harmful to performance, in all the experiments that follow we simply leave k at the conservative default value of 100, unless stated otherwise. Once we fixed F and k we tuned the penalty term γ by testing $\gamma \in \{0.0001, 0.001, 0.01, 0.1\}$. We can see from



(a) Mean Squared Error on the test set as a function of the expected number of message-passing steps, for different values of the penalty term γ . Error bars represent the standard deviation across seeds.

(b) Halting probability distribution over the message-passing steps for different values of the penalty term γ . Shaded areas represent the standard deviation across seeds.

Figure 3.4: Effect of the penalty term γ on the halting mechanism, and on the performance of the model in terms of Mean Squared Error.

Figure 3.4a that a higher value of γ leads to a higher MSE and to a halting probability distribution that is more concentrated on the first steps. This is consistent with the fact that a higher penalty term encourages the model to halt earlier, which can lead to underfitting and worse performance. We choose $\gamma = 0.001$ since it provides the same performance in terms of MSE as $\gamma = 0.0001$ but with less expected message-passing steps, as can be seen in Figure 3.4a. Moreover a closer look at the distribution of

halting probabilities in Figure 3.4b reveals that $\gamma = 0.001$ leads to a halting distribution that goes to zero after step five, while $\gamma = 0.0001$ leads to a halting distribution that is more spread out and has a non-negligible probability of halting at later steps. This suggests that $\gamma = 0.001$ is more effective in preventing oversmoothing.

3.5 Tuning Results

The OAT procedure is inherently limited, as it ignores possible interactions between hyperparameters. Anyhow, even at this level of approximation some clear patterns emerge. Most hyperparameters exhibit wide plateaus of nearly equivalent performance: for instance, the MSE is essentially flat for $k \gtrsim 8$, and the sensitivity to γ is confined to a narrow range of clearly suboptimal values. This is sufficient to get a reliable idea of how to select a configuration that avoid these regions of suboptimality. A more sophisticated tuning procedure [1] for our hyperparameters would likely yield a more precise configuration and possibly marginal performance gains, and is left for future work. The results of the tuning procedure are summarized in Table 3.1.

Hyperparameter	Value
c	10
γ	0.001
F	256
k	100
η	0.001

Table 3.1: Summary of the results of the tuning process. If not stated otherwise, these parameters will be used in the next experiments.

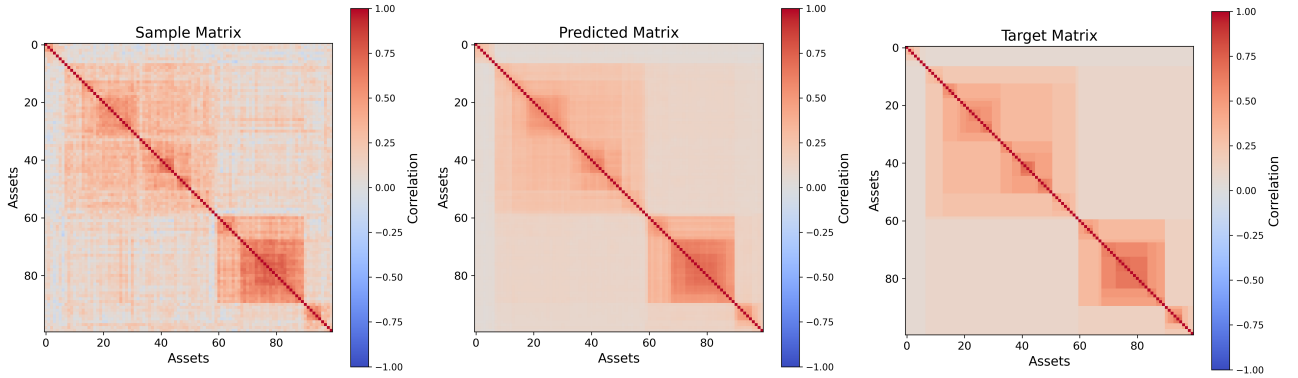
Chapter 4

Results

In this chapter, we first present the results obtained on synthetic datasets (Section 4.1), using this framework to analyze the model’s generalization capabilities. In the second part of the chapter, we transition to the preliminary results obtained on real datasets (Section 4.2), including comparisons with state-of-the-art methods. Finally, we provide examples of real-world applications, with a specific focus on portfolio optimization in Section 4.3.

4.1 Results on Synthetic Datasets

We begin our analysis by evaluating the model on synthetic datasets. Because these datasets allow for complete control over the underlying ground-truth structure, they provide a clear picture of the model’s capabilities before we transition to real-world data. The datasets we use to assess the performance of our model are those described in Section 3.1. In Figure 4.1 we can see an example of a noisy input correlation matrix $\hat{C}_{Sample}$ and the corresponding predicted matrix $\hat{C}$ obtained using the GNN tuned in the previous section. In Figure 4.1c we can see the target correlation matrix C corresponding to the



(a) Example of input matrix $\hat{C}_{Sample}$ (b) Example of predicted matrix $\hat{C}$ (c) Example of target matrix C

Figure 4.1: Example of a denoising task from the base dataset.

input matrix $\hat{C}_{Sample}$ in Figure 4.1a. It is worth noting that the predicted matrix $\hat{C}$ in Figure 4.1b is able to capture the mesoscopic structure of the target matrix C in Figure 4.1c, showing that the GNN is able to effectively denoise the input matrix $\hat{C}_{Sample}$ and recover the underlying correlation structure. This is precisely what a Rotationally Invariant Estimator cannot achieve. Being confined to the sample eigenvectors, an RIE can rescale the spectrum but cannot recover the cluster organization of the market, which is encoded in the eigenvectors.

Thanks to our architecture we can also inspect what the GNN is doing at each layer. Figure 4.2 shows, for this representative test sample, two complementary views at three steps. The top row

reports the *incremental* reconstruction $\hat{C}^{(\leq s)}$ accumulated up to step s (the running, halting-weighted estimate), while the bottom row shows the raw *single-step* reconstruction $\hat{C}^{(s)}$. The structure emerges from local to global. At the first step (Fig. 4.2a) only the tightest clusters are resolved; by the intermediate step (Fig. 4.2b) the mid-level communities appear; and at the final step (Fig. 4.2c) the full top-level partition is recovered. The nested block hierarchy is clearly visible at all scales. We can see how, by step 8 (Figure 4.2f), the matrix collapses into an almost uniform fully correlated block. The controlled, halting-weighted combination allows for a certain amount of oversmoothing; the right amount to recover the global market mode and the top-level partition, but not so much that the multi-scale block structure is washed out into a single oversmoothed block. We can see the halting weights distribution in Figure 4.3.

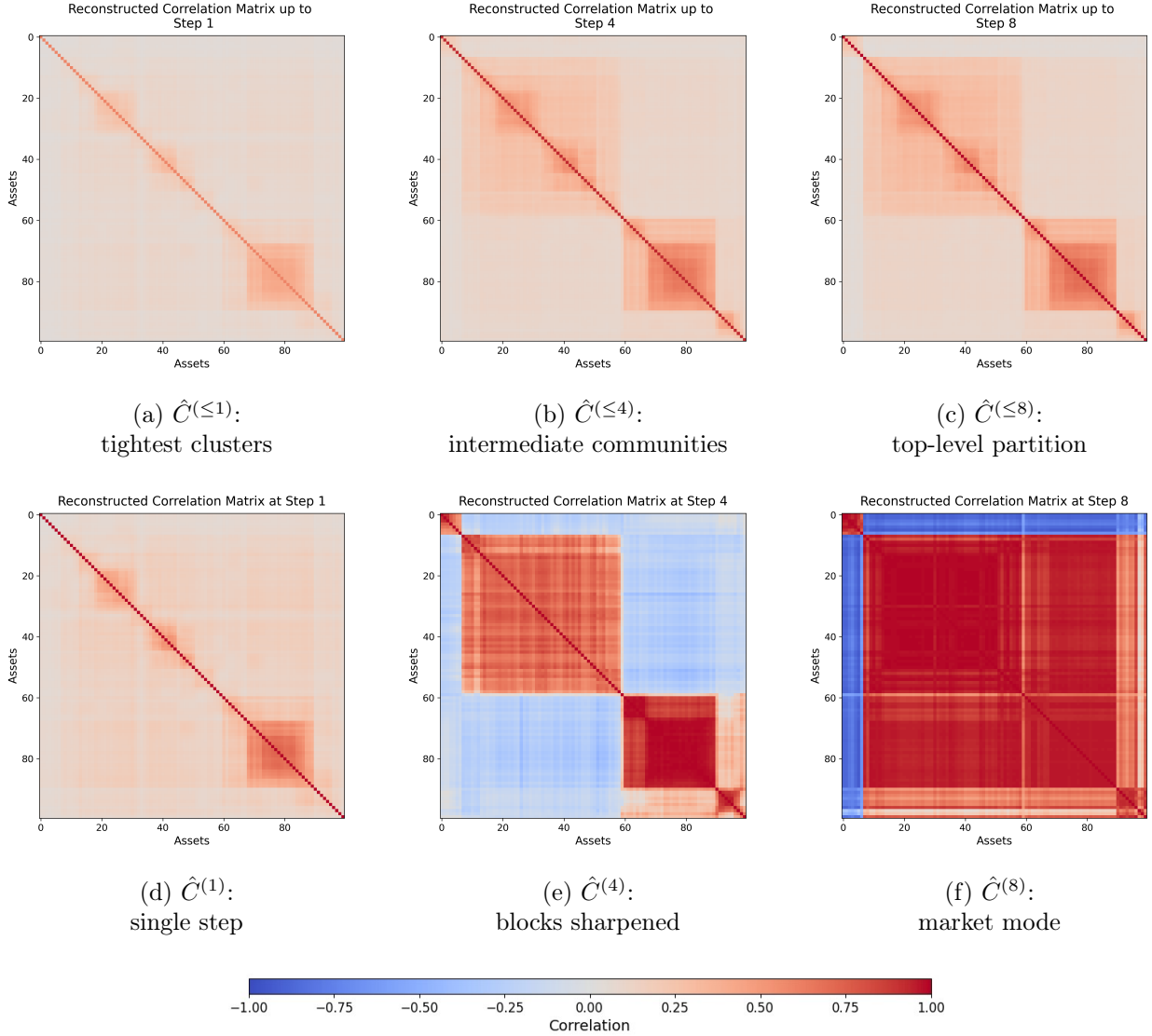


Figure 4.2: Layer-wise reconstruction of the GNN. **Top row:** $\hat{C}^{(\leq s)}$ reconstructed up to step s . **Bottom row:** $\hat{C}^{(s)}$ decoder output at steps 1, 4 and 8. The structure builds up from local clusters to the global market mode as the steps progress.

Now that we have seen this example, we can move on to more quantitative results.

To validate the performance of our model, we compared our results with the BAHG estimator [4, 2], which has been shown to be effective in similar contexts. We need to be careful in the comparison: the eigenvectors of the BAHG estimator are not the same as the ones of the target matrix, and the same is true for the eigenvectors of the predicted matrix $\hat{C}$. So for each of them we will project the target into the predicted matrix eigenvectors'. For each estimator we will report the MSE_{eig} , MSE_{vec} , MSE .

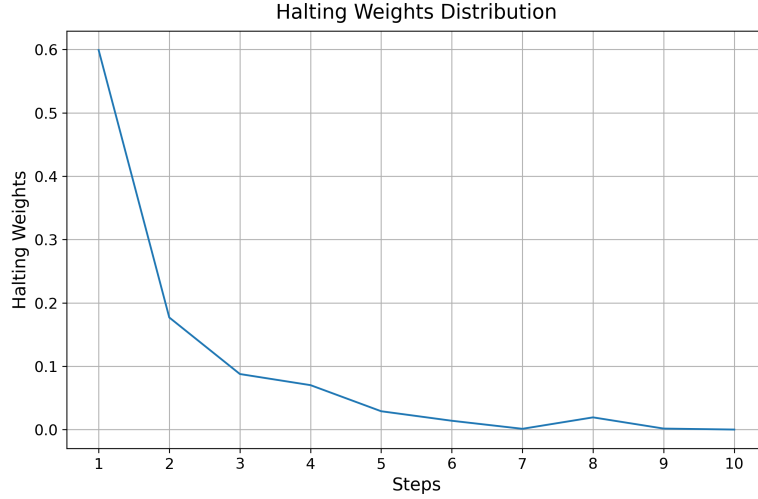


Figure 4.3: Distribution of halting weights.

The results of the comparison for our previous example are reported in Table 4.1.

	MSE_{eig}	MSE_{vec}	MSE (total)
sample	0.00180	0.00465	0.00644
GNN	0.00014	0.00088	0.00102
BAHC	0.00024	0.00072	0.00096

Table 4.1: MSE decomposition into eigenvalue and eigenvector contributions for the sample correlation, our reconstruction (*GNN*) and the BAHC benchmark. Lower is better; the best value in each column is in bold.

For the sake of completeness, we can also project the predicted matrix $\hat{C}$ and the BAHC estimator onto the target eigenvectors U , and compare the diagonal elements of the projected matrices with the target eigenvalues Λ . The results are reported in Figure 4.4. Projecting the cleaned matrices onto the target eigenbasis shows us that the variance captured along each true direction closely follows the target eigenvalues. This suggests that the directions of the eigenvectors of the target matrix C are well approximated by the eigenvectors of our predicted matrix $\hat{C}$ and the BAHC estimator. This should be supported by the fact that the off-diagonal elements of the projected matrices are smaller than in the sample matrix. In Figure 4.5 we can see the comparison of the spectra of our predicted matrix $\hat{C}$, the BAHC estimator and the target matrix C . We can also see the spectrum of the sample correlation matrix $\hat{C}_{\text{Sample}}$, which is very noisy. This shows that our model is able to clean the spectrum of the input matrix $\hat{C}_{\text{Sample}}$ from the noise as effectively as the BAHC estimator does.

Beyond this illustrative example, we now report the results obtained on the whole test dataset of 500 samples. The results presented in Table 4.2 show that our model is able to solve the denoising problem

Method	MSE $\pm$ Std Error	95% CI
Sample	0.0051 $\pm$ 0.0001	[0.0050, 0.0053]
GNN	0.0012 $\pm$ 0.0001	[0.0012, 0.0014]
BAHC	0.0013 $\pm$ 0.0001	[0.0012, 0.0014]

Table 4.2: Comparison of the Mean Squared Error (MSE) estimated via bootstrap. Bold values indicate the best performance (lowest mean error).

on the base dataset, achieving performance comparable to the BAHC estimator. In C we report the results obtained on the other synthetic datasets, showing that our model is able to generalize well.

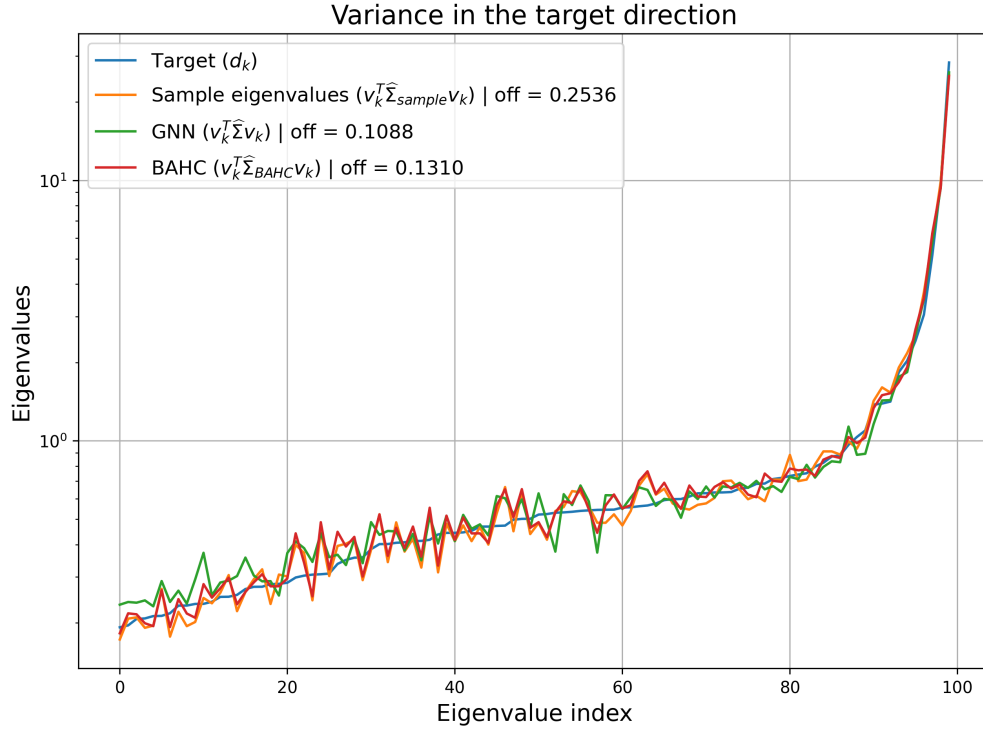


Figure 4.4: Comparison of the diagonal elements of our estimators projected onto the target basis with the target eigenvalues Λ . In the legend we report the magnitude of the off-diagonal elements of the projected matrices

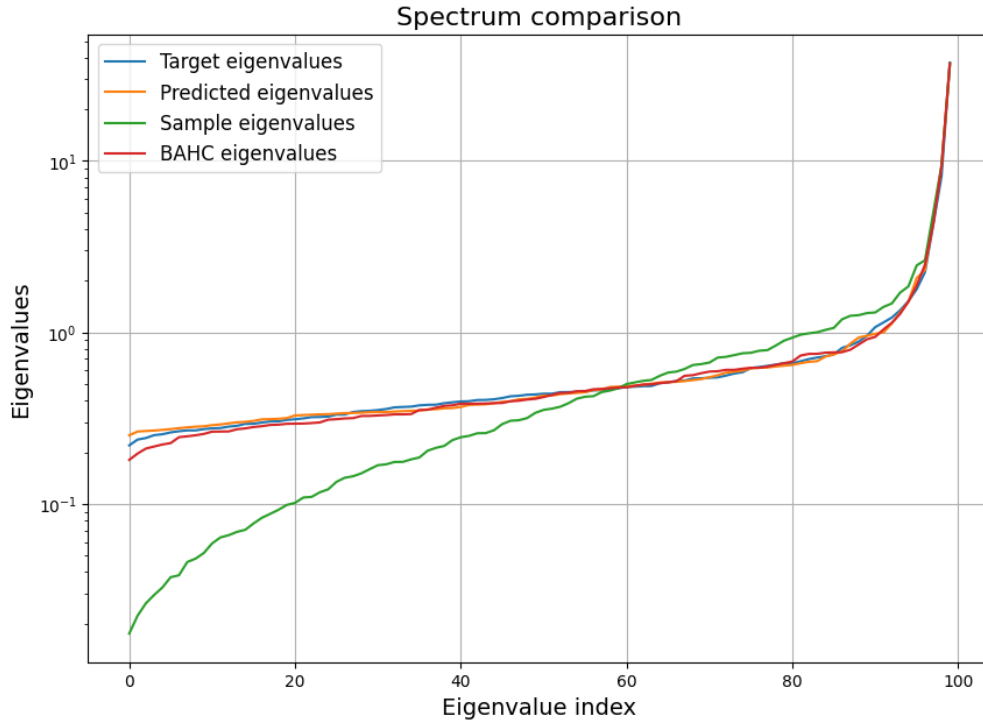


Figure 4.5: Comparison of the different spectra

4.1.1 Scaling With the Matrix Size

An important aspect we need to consider is how the performance and the dynamics of our model change when we increase the size of the input matrix. To investigate this we trained our model on the base dataset using matrices of size $N \in \{50, 100, 150, 200\}$. Furthermore, since our architecture

accommodates varying matrix dimensions, we can evaluate its generalization capabilities by testing a model trained on matrices of size N on unseen sizes ($N' \neq N$). This also allows us to investigate the impact of providing the ratio q as a global attribute. It is evident from Figure 4.6 that each model

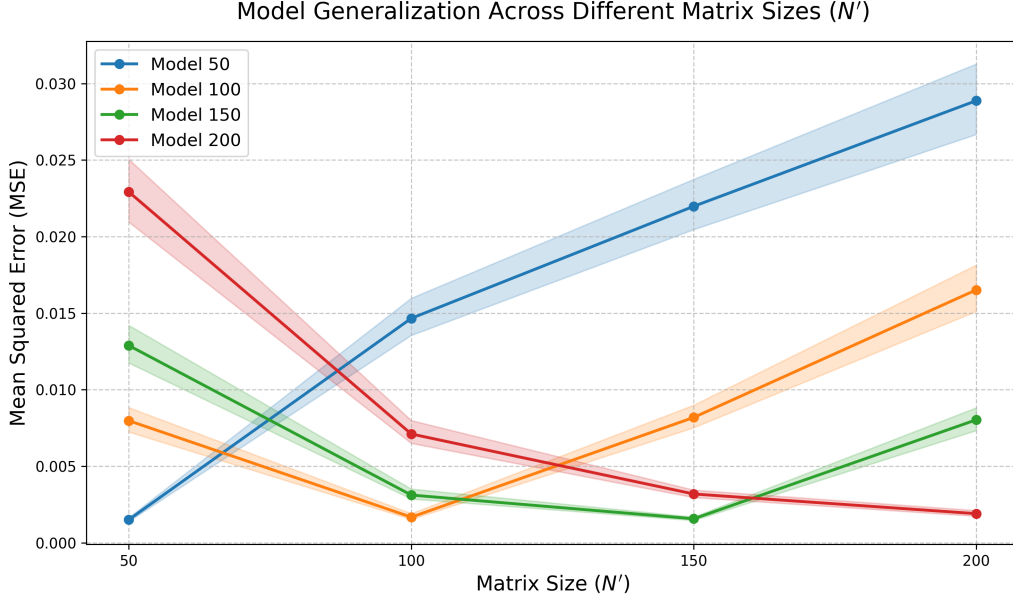


Figure 4.6: Mean Squared Error (MSE) as function of the matrix size N . The model is trained on matrices of size N and tested on unseen sizes N' . The plot displays the Mean Squared Error (MSE) along with 95% confidence intervals, estimated via bootstrap.

achieves its lowest reconstruction error when evaluated on its native training size, with performance degrading as the test dimension deviates from the training dimension. Despite this, analyzing the internal dynamics of the model, we observe that this seems to be independent of the matrix size, as shown in Figure 4.7.

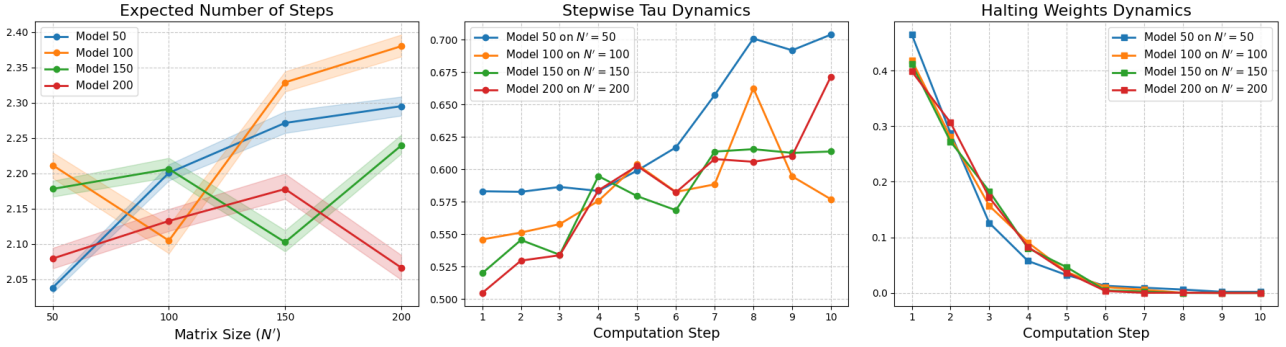


Figure 4.7: From left to right the figures show the expected number of steps, the dynamics of the parameter τ and the distribution of halting probabilities for the different models.

These results suggest that a more efficient training strategy could be to train a single model on matrices of different sizes and provide the ratio q as a global attribute, allowing the model to generalize across different matrix dimensions. We trained a model on matrices of size $N = 100$ and another model on matrices of sizes $N \in [80, 120]$ and we tested both models on matrices of size $N \in \{50, 100, 150, 200\}$ and evaluated their performance in terms of MSE with respect to the target matrix. The results are reported in Figure 4.8.

We can see that the performance of the model trained on matrices of different sizes is roughly constant across the whole testing range, while the model trained on matrices of size $N = 100$ reaches its lowest reconstruction error when evaluated on its native training size, with performance degrading as the test

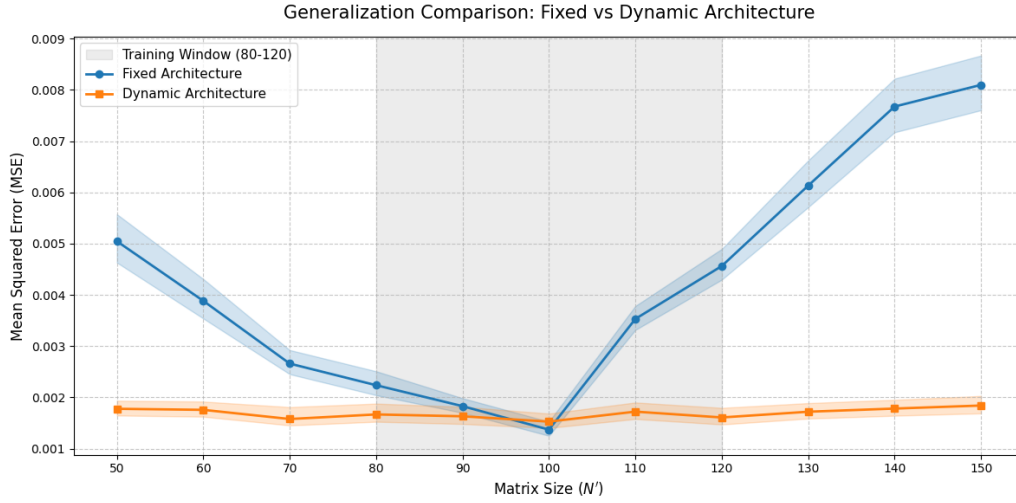


Figure 4.8: Generalization capabilities of the model trained on matrices of size $N = 100$ and the model trained on matrices of size $N \in [80, 120]$. The plot displays the Mean Squared Error (MSE) along with 95% confidence intervals, estimated via bootstrap.

dimension deviates from the training dimension. This confirms that providing the ratio q as a global attribute and training the model on different matrix sizes is sufficient to allow the model to generalize across different matrix dimensions.

4.1.2 Scaling With the Length of the Time Series

When dealing with noisy data, the length of the time series T used to compute the sample correlation matrix $\hat{C}_{Sample}$ is a crucial factor. A natural question that arises is how well our model performs under different T . We therefore trained our model on the base dataset using matrices of size $N = 100$ and varying the length of the time series $T \in [100, 300]$. Then we trained another model on matrices of the same size but with fixed $T = 150$. We compared the performance of the two models on matrices of size $N = 100$ and varying the length of the time series $T \in [50, 400]$. Results are reported in Figure 4.9. It

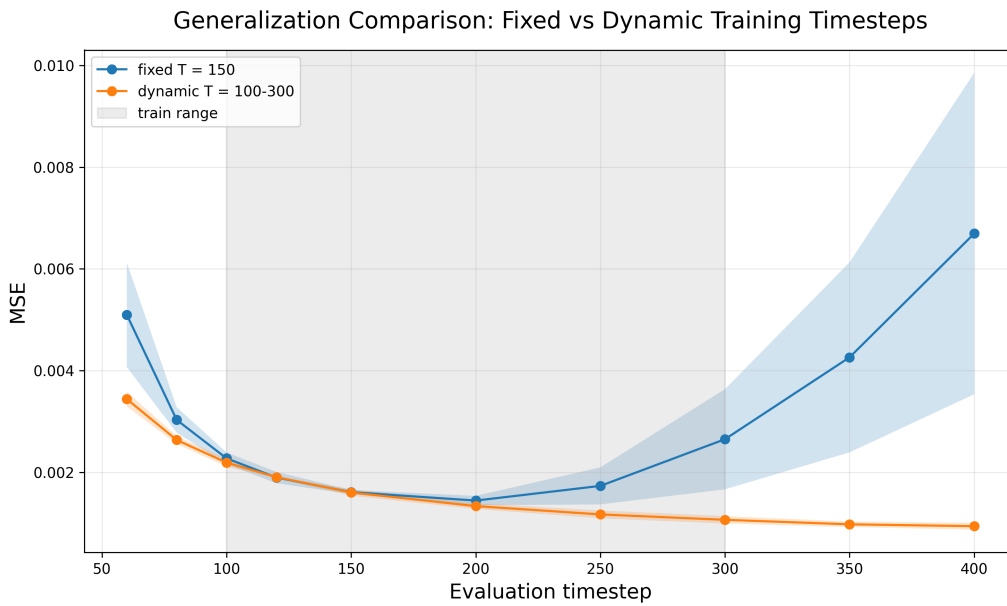


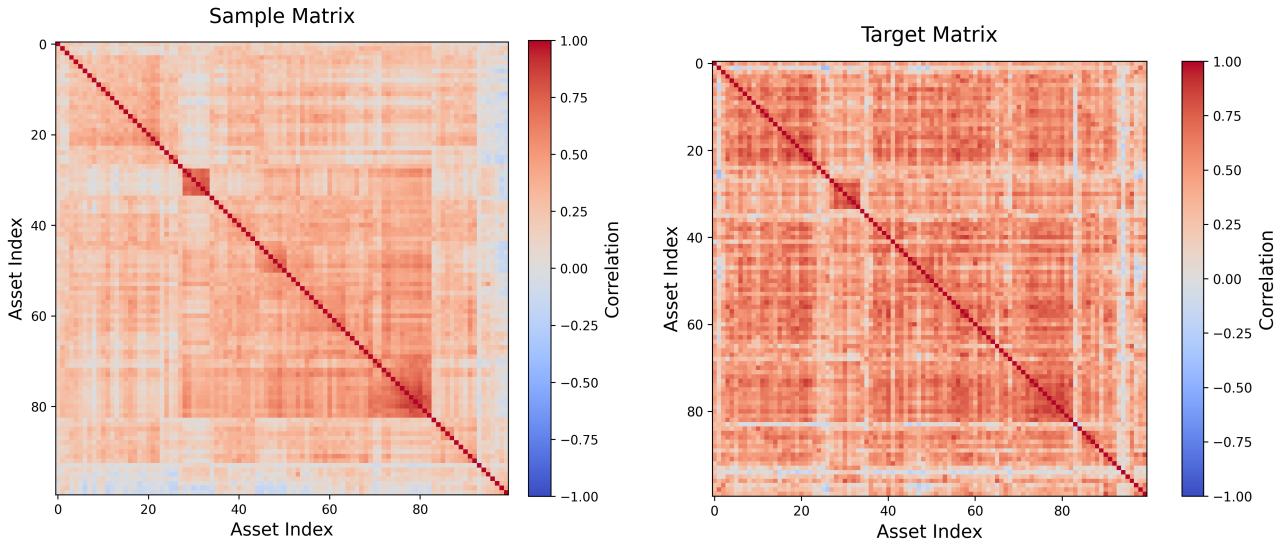
Figure 4.9: Generalization capabilities of the model trained on matrices of size $N = 100$ and $T = 150$, and the model trained on matrices of size $N = 100$ and $T \in [100, 300]$. The plot displays the Mean Squared Error (MSE) along with 95% confidence intervals, estimated via bootstrap.

is not surprising that we obtain a trend similar to the one observed when varying the matrix size N , indeed from Random Matrix Theory [10] we know that the relevant parameter is the ratio $q = \frac{N}{T}$.

In the end, training a model with N and T kept constant means that the network is trained to handle a specific amount of noise determined by the ratio q . Instead, if during training we vary q , by changing N or T or both, the model will be able to understand the level of noise in the input matrix and adapt its dynamics accordingly. We can also observe that as the ratio q decreases (i.e., as T increases relative to N), the performance of the dynamic model improves, as expected since the sample correlation matrix becomes less noisy.

4.2 Results on Real Dataset

Having established how our model behaves on synthetic datasets, we now transition to real-world data. We train our model on correlation matrices of 100 assets from US equities with high market capitalization from 01/01/2000 to 31/12/2025. We use a window of 300 days as input, and as target the realized correlation matrix computed over the following 30 days. This setup implicitly assumes that the correlation structure is approximately stationary over the combined input–target horizon, so that the past 300 days are informative about the following 30. During training, we compute the validation loss using as target correlation matrices computed over a window that can be larger or smaller than the test one, depending on the intended application.



(a) Example of a real input matrix $\hat{C}_{Sample}$

(b) Example of a real target matrix C

Figure 4.10: Example of a pair of sample and target correlation matrices from the real dataset. In order to make the structure evident, we have permuted the rows and columns of the two matrices using Ward linkage.

In Figure 4.10 we can see an example of a pair of sample and target correlation matrices from the real dataset. As done before we will first compare our results for a single sample and then we will move to the results obtained on the whole test dataset. We will compare our results with the BAHG estimator, with the QIS estimator [19], and with a naive estimator obtained by clipping the eigenvalues [8]. In Figure 4.11 we can see the cleaned correlation matrices of Figure 4.10a using our GNN model and the QIS estimator. For the sake of completeness, in D we report the results obtained using the BAHG estimator and the naive clipping estimator in Figure D.1. As done for the synthetic dataset, we can have a look at the eigenvalues of the different estimators as function of the rank. It is helpful to see this in order to understand how the different estimators act on the spectrum of the input matrix $\hat{C}_{Sample}$. The results are reported in Figure 4.12.

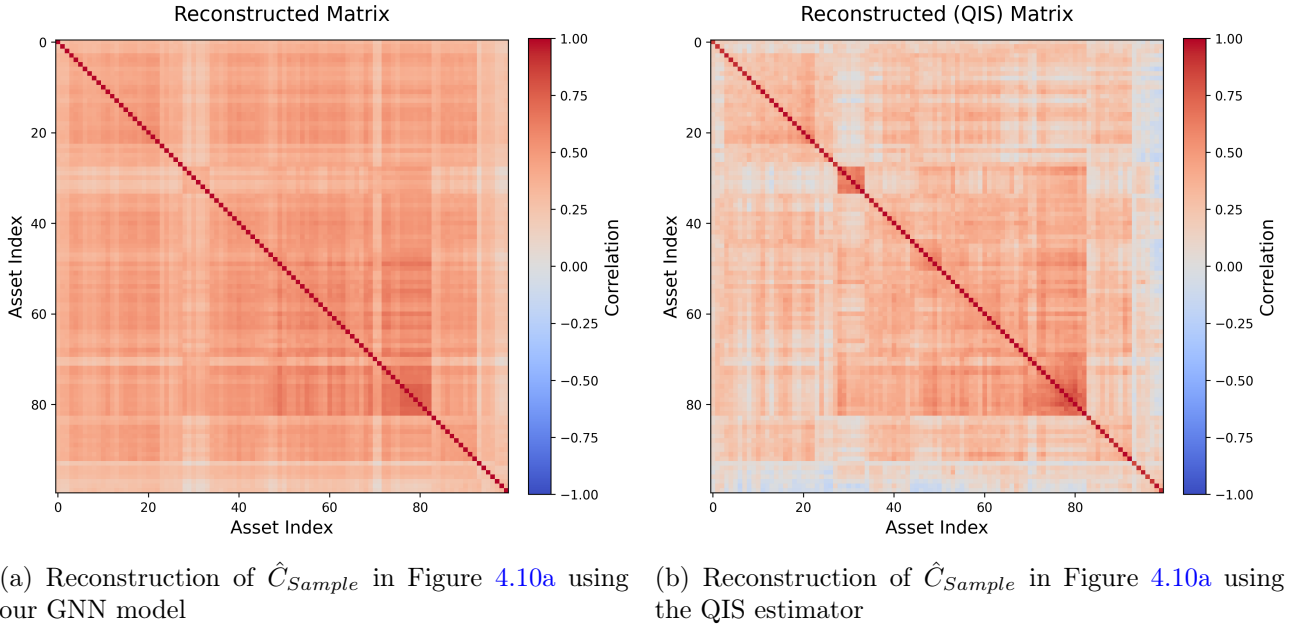


Figure 4.11: Example of cleaned correlation matrices of Figure 4.10a using our GNN model and the QIS estimator. In order to make the structure evident, we have permuted the rows and columns of the two matrices using Ward linkage.

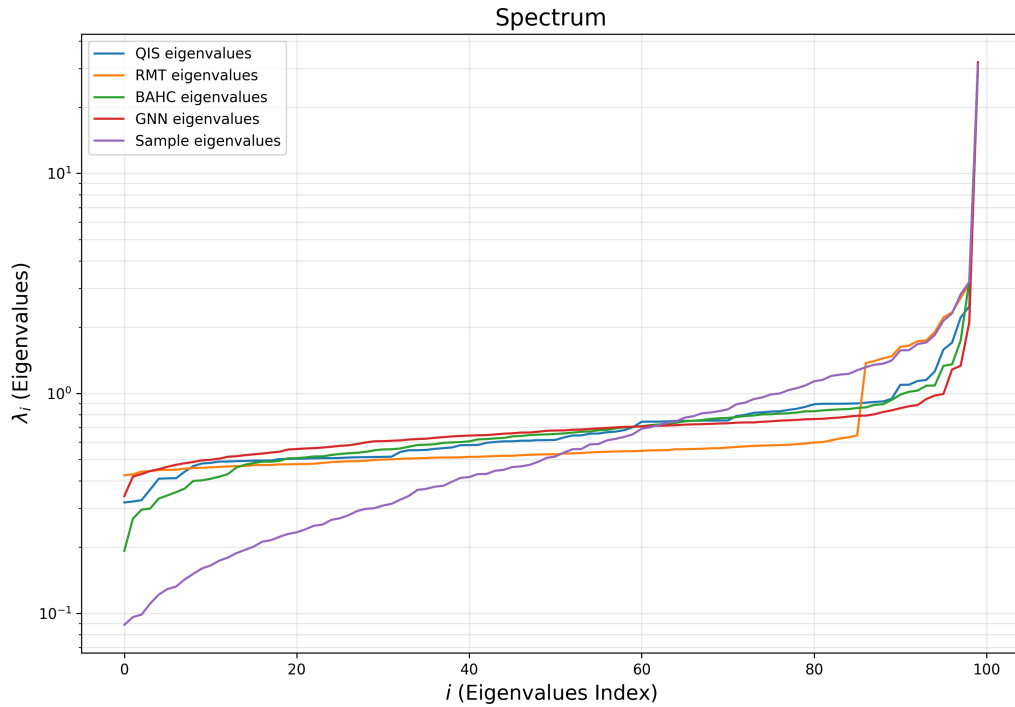


Figure 4.12: Comparison of the eigenvalues of the different cleaned matrices in the basis of the sample correlation matrix $\hat{C}_{Sample}$.

It is important to note that the target matrix C is computed over a window of 30 days; this means that it is very noisy and not even invertible, having 71 eigenvalues equal to zero. We used this setup because in real applications we are interested in rebalancing our portfolio on a short time horizon. Our aim is to capture the persistent structure of the correlation matrix, which is expected to be stable over time, while filtering out the noise.

Having discussed this illustrative example, we now turn to the results obtained on the whole test dataset of 500 samples. The results are reported in Table 4.3. To complement the absolute mean squared error,

Estimator	MSE $\pm$ Std Error	95% CI
Sample	0.043957 $\pm$ 0.023425	[0.041931, 0.046114]
QIS	0.042400 $\pm$ 0.023546	[0.040460, 0.044842]
RMT	0.043380 $\pm$ 0.023393	[0.041193, 0.045730]
GNN	0.041779 $\pm$ 0.021594	[0.040105, 0.044231]
BAHC	0.043193 $\pm$ 0.023274	[0.041301, 0.045468]

Table 4.3: Mean squared error (MSE) of each correlation-matrix estimator against the out-of-sample target. The values are estimated via bootstrap, the 95% confidence interval (CI) is reported. The best (lowest) value is shown in bold.

we report the Percentage Relative Improvement in Average Loss (PRIAL), which measures the fraction of the sample estimator’s loss removed by each method,

$$\text{PRIAL}(\hat{C}) = \left(1 - \frac{\mathbb{E}[L(\hat{C})]}{\mathbb{E}[L(\hat{C}_{\text{Sample}})]} \right) \times 100 \%, \quad (4.1)$$

where L is the out-of-sample MSE and $\hat{C}_{\text{Sample}}$ is the raw sample correlation matrix. The PRIAL is zero when an estimator matches the sample benchmark and approaches 100% as the loss vanishes, providing a scale-free measure of improvement over the naïve estimator. As shown in Tables 4.3 and 4.4, the classical estimators (QIS, RMT, BAHC) yield only modest gains (PRIAL at most about 3.5%), whereas the GNN removes close to 5% of the sample loss. This advantage is in part expected, since the network is explicitly trained to minimise the same MSE against the out-of-sample target, while the other estimators optimise different statistical criteria; the comparison on this metric is therefore favourable to the learned model by construction.

Estimator	PRIAL (%)	95% CI
QIS	3.543	[2.940, 4.197]
RMT	1.315	[1.141, 1.563]
GNN	4.957	[0.987, 9.732]
BAHC	1.738	[1.382, 2.063]

Table 4.4: Percentage Relative Improvement in Average Loss (PRIAL) of each estimator with respect to the sample correlation matrix. Positive values indicate a reduction in MSE relative to the sample estimator. The values are estimated via bootstrap, the 95% confidence interval is reported. The best (highest) value is shown in bold.

In any case we need to be careful in interpreting these results. This setup is different from what we have done in the synthetic case, since there we had a true stationary target correlation to reconstruct. Here, although the scenario is more realistic, it is also considerably more extreme. Further experiments have confirmed that by using a more stable target, calculated over the 300 days following the sample window, we obtain an even better result in terms of MSE (roughly halved). The comparison itself remains unchanged, with the GNN outperforming the competing estimators. The GNN attains a PRIAL of 9.46%, compared with 4.86% for the QIS estimator. However, the real performance of the model should be evaluated in terms of portfolio optimization, which is the ultimate goal of this work.

4.3 Portfolio Optimization

Moving beyond statistical accuracy, we now evaluate the model’s utility in a practical financial setting. Achieving a lower MSE against a noisy 30-day target correlation matrix is an encouraging result.

However, as demonstrated in Ref. [3], minimizing the MSE does not automatically translate to superior out-of-sample portfolio performance.

In the following we will analyze the case of the Global Minimum Variance portfolio (GMV). In order to evaluate the performance of the different covariance estimators, let us introduce the setting we use and the metrics we will use to compare the different portfolios. We consider a single, continuously managed portfolio of $N = 100$ assets. At each rebalancing date the investable universe consists of the 100 most capitalized stocks at that date. We allow for both long and short positions, and we do not consider transaction costs. The long-only case, by contrast, admits no closed-form solution and requires numerical optimization. Results for the long-only case are reported in Appendix E. We looked at the period from 01/01/2000 to 31/12/2025. We used a rolling window of 300 days to estimate the covariance matrix. Specifically, at each rebalancing date we used the most recent 300 daily observations to compute the sample correlation matrix $\hat{C}_{Sample}$ and, through the different covariance estimators, the covariance matrix $\hat{\Sigma}$. The resulting Global Minimum Variance (GMV) weights were then held fixed and used to allocate the portfolio over the following 30 days. At the next rebalancing date the window is shifted forward by 30 days. In this way there is no overlap between in-sample and out-of-sample data, and the out-of-sample performance of the different covariance estimators can be evaluated.

Another important aspect to take into account is the drift of the weights of the portfolio. Once we have predicted the optimal weights, we build the portfolio and we hold it for 30 days. The problem is that, over time, asset prices change and as a consequence the available capital invested in each asset changes as well. This clearly changes the composition of the portfolio, i.e. the weights of the individual assets; this is called drift. If we hold for a time period T we can define the value of the asset i at time t as:

$$V_i(t) = w_i \prod_{j=1}^t (1 + r_i(j)) \quad (4.2)$$

where w_i is the initial weight of the asset i and $r_i(j)$ is the return of the asset i at time j . The product in equation 4.2 represents the growth factor of the asset i over the period of time t , we can call it $G_i(t)$. The drifted weight of the asset i at time t can be defined as:

$$w_i(t) = \frac{V_i(t)}{\sum_{j=1}^N V_j(t)} = \frac{w_i G_i(t)}{\sum_{j=1}^N w_j G_j(t)} \quad (4.3)$$

So if we want to know the return of the portfolio at time $t + 1$ we can compute it as:

$$r_p(t + 1) = \sum_{i=1}^N w_i(t) r_i(t + 1) \quad (4.4)$$

One can show (B) that the return of the portfolio at time T can be computed as:

$$R_p(T) = \sum_{i=1}^N w_i^{(0)} (G_i(T) - 1) \quad (4.5)$$

In Figure 4.13 we can see the cumulative returns of the GMV portfolio constructed using each covariance estimator. We see that the GNN-based portfolio shows the highest cumulative return over the evaluation period. Other important metrics to consider are the volatility, the Sharpe ratio and the average turnover. The volatility σ_p is computed as the standard deviation of the portfolio returns over the evaluation period, while the Sharpe ratio is computed as:

$$\text{Sharpe} = \frac{\mathbb{E}[R_p]}{\sigma_p} \quad (4.6)$$

The average turnover is computed as the average of the absolute difference between the weights of the portfolio at time t and the weights of the portfolio at time $t - 1$. It is a measure of how much the portfolio is rebalanced over time, and it is an important metric to consider since it can have a significant

impact on the transaction costs. To quantify portfolio concentration, the Herfindahl-Hirschman Index (HHI) is a standard metric defined as the sum of the squared portfolio weights. For a portfolio with N assets and weights w_i , it is calculated as:

$$HHI = \sum_{i=1}^N w_i^2 \quad (4.7)$$

While HHI values range from $\frac{1}{N}$ (perfect diversification) to 1 (maximum concentration), the raw score can be unintuitive. Therefore, quantitative practitioners often use the Effective Number of Constituents (ENC), which is simply the reciprocal of the HHI:

$$ENC = \frac{1}{HHI} \quad (4.8)$$

The ENC translates the concentration risk into a more readable format, indicating the equivalent number of equally-weighted assets that would produce the same level of portfolio concentration. All the metrics are reported in Table 4.5. As is common in the literature, we report the annualized return and volatility, the Sharpe ratio, the cumulative return and the average turnover. We also report the annualized volatility for each year in Figure 4.14, for each covariance estimator.

Method	Ann. Return	Ann. Vol.	Sharpe	Cumulative	Turnover	N_{eff}
Sample	0.0467	0.1425	0.328	1.5714	12.2135	53.79
QIS	0.0634	0.1290	0.491	3.1411	6.5595	50.10
RMT	0.0686	0.1339	0.512	3.6626	7.6797	50.42
GNN	0.0865	0.1469	0.589	6.0706	5.6348	46.42
BAHC	0.0670	0.1398	0.479	3.3882	5.0530	49.20
EWP (<i>benchmark</i>)	0.0899	0.1939	0.464	5.2704	0.7571	99.62

Table 4.5: Out-of-sample portfolio performance for each covariance estimator, including the equally weighted portfolio (EWP) as a benchmark: annualised return, annualised volatility, Sharpe ratio, cumulative return, average turnover, and effective number of constituents N_{eff} . The best value in each column is shown in bold (highest return, Sharpe, and cumulative return; lowest volatility and turnover).

From these results we can see that the GNN-based portfolio shows the highest Sharpe ratio and cumulative return over the evaluation period. Its average turnover is comparable with the lowest one, obtained by BAHC, which translates into lower transaction costs. The volatility of the GNN-based portfolio is slightly higher than that of the other portfolios, but this is compensated for by the higher return. In the end, the Sharpe ratio is higher for the GNN-based portfolio. This means that the GNN-based portfolio is able to achieve a better risk-adjusted return compared to the other portfolios. If we look at the Equally Weighted Portfolio (EWP) as a baseline, we can see that despite the fact that it has the highest return, it also has the highest volatility and lowest Sharpe ratio. Clearly the EWP has the lowest turnover, since it is driven only by the drift of the weights. It is worth underlining that the use of a different loss function could lead to different results. In particular, a loss function that explicitly accounts for the portfolio variance would likely yield better results on that metric.

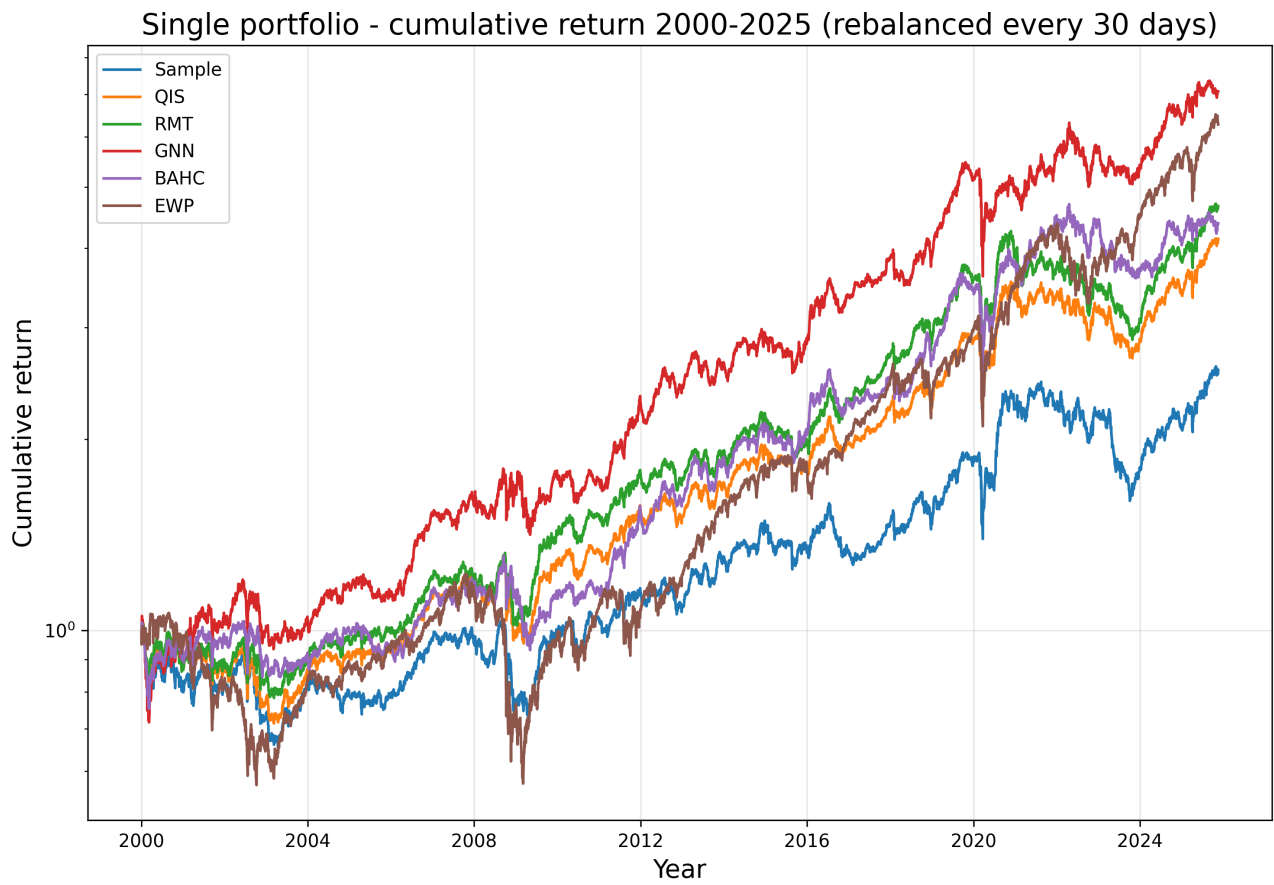


Figure 4.13: Cumulative returns of the GMV portfolio constructed using each covariance estimator. The GNN-based portfolio shows the highest cumulative return over the evaluation period.

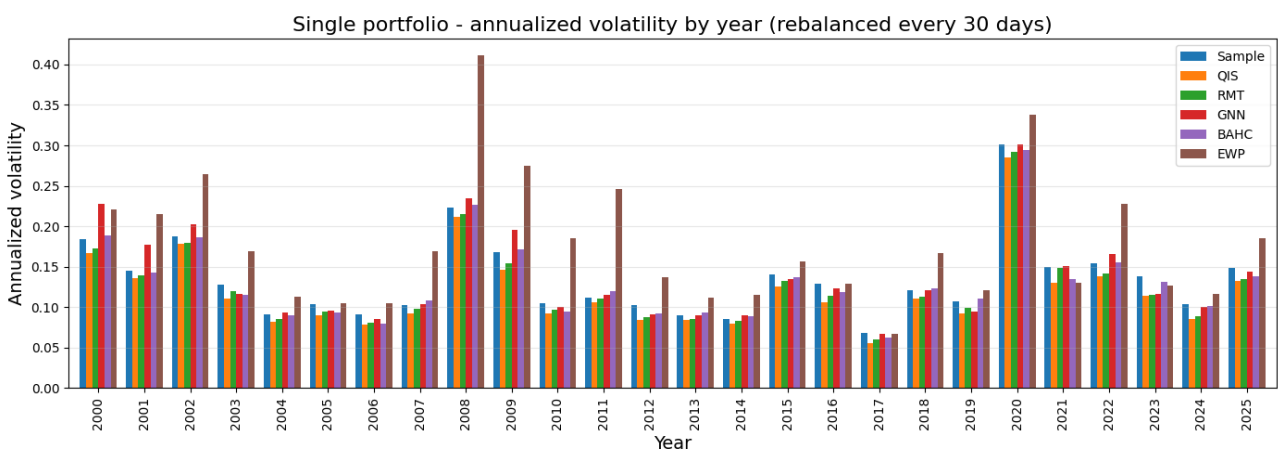


Figure 4.14: Annualised volatility of the GMV portfolio constructed using each covariance estimator.

Chapter 5

Conclusions

This final chapter summarizes the core findings of our work and discusses the limitations of the proposed architecture. First, in Section 5.1, we investigate scenarios where the Graph Neural Network struggles to match theoretical baselines. Finally, Section 5.2 draws the overarching conclusions of this study and outlines potential perspectives for future research.

5.1 A Unified Approach

A natural question, recurring throughout this work, is whether the GNN can reproduce, in a fully controlled setting, the optimal cleaning prescribed by Random Matrix Theory. We isolate the hardest possible case for our GNN: a white inverse-Wishart input. The theory predicts that the optimal rotationally invariant estimator reduces to a linear shrinkage of the sample matrix towards the identity [5], with a shrinkage intensity known in closed form. Since an inverse-Wishart matrix carries no eigenvector structure, there is nothing for the network to exploit. To quantify how well each estimator recovers this operation, we regress the off-diagonal entries of the cleaned matrix onto those of the sample matrix, the slope β is the effective shrinkage factor ($\beta = 1$ leaves the sample untouched, $\beta = 0$ collapses to the identity). Table 5.1 reports β for the plain GNN trained with the MSE loss, for QIS, and for the theoretical target.

	$\beta \pm$ Std Error	95% CI
Theoretical shrinkage factor	0.6024	—
GNN (MSE loss)	0.4155 ± 0.0026	[0.4104, 0.4205]
QIS	0.5817 ± 0.0022	[0.5774, 0.5861]

Table 5.1: Shrinkage coefficient β estimated via linear regression on the off-diagonal elements, for a white inverse-Wishart input. Standard deviation and 95% confidence intervals are obtained by bootstrapping over 500 matrices. The theoretical shrinkage factor is reported as a reference.

The outcome is informative rather than a failure. The network does learn to shrink: it is far from the trivial $\beta \approx 1$, but it over-shrinks, driving the off-diagonals too aggressively towards zero ($\beta = 0.4155$ against a target of 0.6024). Two mechanisms explain this bias. First, the architecture is not rotationally invariant by construction, and message passing is designed to aggregate local neighbourhood information; a purely global operation, identical for every entry, is precisely what such a scheme is least equipped to represent. Second, on a structureless target the MSE objective is minimised, on average, by regressing towards the mean, i.e. towards the identity, which manifests as systematic over-shrinkage. QIS, being a genuine RIE tailored to this setting, sits essentially on the theoretical value. Crucially, this shortcoming is confined to the pathological, structureless corner. On the mix benchmark, where

an inverse-Wishart contamination is superimposed on a hierarchical structure that is always present (Table C.2), the model performs excellently. There the network can do what it was designed for: recover the informative structures. The controlled experiment therefore delimits the boundary of the approach rather than invalidating it. The GNN is the right tool when there is structure to exploit, and a suboptimal one when the signal is pure noise.

This diagnosis points directly to two remedies, both of which recover a near-optimal shrinkage factor. Replacing the MSE with an information-theoretic objective (the KLD/Stein loss, (Appendix G) removes the over-shrinkage bias by asymmetrically penalising the under-estimation of the variance directions, raising β to 0.5658. Alternatively, a residual-learning scheme (Appendix F) delegates the denoising of the structureless residual to a dedicated rotationally invariant network (RIEnet), which brings β to 0.5827, in statistical agreement with both the theoretical value and QIS. These results suggest that an effective way of using this GNN is as a module within an end-to-end framework trained to minimize the portfolio variance, where each block is responsible for a specific task, such as estimating the correct variances, cleaning the structure, and cleaning the spectrum. This is clearly a complex task that requires time and the careful design of a structured pipeline, but the experiment in Appendix F suggests that it is a direction worth pursuing.

5.2 Conclusion and Future Work

This work has shown that GNNs can be used to denoise covariance matrices and that most importantly, they can be used to extract the underlying structure of the covariance matrix. We designed a Graph Neural Network that represents assets as nodes and pairwise dependencies as edges, and that combines message passing with a differentiable halting mechanism. The halting branch turns oversmoothing into a resource. We have seen how early iterations resolve the tightest clusters, intermediate ones the mesoscopic communities, and the final ones the global market mode. The halting-weighted combination of the per-step reconstructions reproduces the hierarchy of the market at every scale. By construction, the decoder always returns a valid correlation matrix.

On synthetic data the model matches the state-of-the-art hierarchical estimator (BAHC) on the base dataset and generalizes to the more demanding mix and overlap settings. On real data, the GNN achieve a lower reconstruction error than QIS, RMT clipping, and BAHC. Moreover, we have shown that GNNs can be used to construct a portfolio that outperforms portfolios constructed using other covariance estimators. In a 25-year out-of-sample backtest of the Global Minimum Variance portfolio the GNN-based allocation attains the highest Sharpe ratio and cumulative return, together with a low average turnover.

However, Section 5.1 delimits the boundaries of validity for this approach. To address these limitations, we proposed two remedies. The emerging picture is that the GNN is the optimal tool when there is an underlying structure to exploit. Several directions follow naturally from these results. On the architectural side, the initial feature encoding is currently learned from scratch, whereas the eigenvectors of the sample matrix are available and would provide a spectrally informed initialization. On the objective side, two refinements appear promising. First, we could introduce a loss that directly penalizes the out-of-sample portfolio variance, aligning training with the financial end goal. Second, we could develop an end-to-end modular pipeline, as suggested by the residual-learning experiment of Appendix F, where specialized blocks handle variance estimation, structural cleaning, and spectral cleaning, trained jointly to minimize portfolio risk. On the empirical side, the present study is limited to the universe of the top 100 capitalized assets, without transaction costs, scaling to larger and more heterogeneous universes, introducing cost-aware rebalancing, and testing robustness across market regimes are necessary steps before any practical deployment.

Beyond the specific estimator proposed here, we believe the main contribution of this work is a proof of concept. It would be interesting to address the major limitation of Rotationally Invariant Estimators, by using GNNs as an exploration tool. Rotationally invariant estimators are limited by the fact

that they completely ignore the eigenvectors of the covariance matrix, and they only focus on the eigenvalues. GNNs can be used to extract the underlying structure of the covariance matrix, and to understand what kind of structures are present, which are static and which are dynamic, and how they evolve over time. We believe that this information can be then exploited to correct the eigenvectors of the covariance matrix, and to construct a better covariance estimator. In this direction, GNNs can be particularly useful, since at each layer they offer a clear picture of what is happening and what kind of structures are being extracted. We regard this work as a starting point for the development of Rotationally Sensitive Estimators (RSE), a new class of estimators in which structure information is used to correct the eigenvectors of the sample covariance matrix.

Appendix A

Synthetic Datasets

In this appendix, we will present the other two kinds of datasets used during the experiments. We first present the so-called 'mix' dataset in Appendix A.1, followed by the overlap dataset in Appendix A.2.

A.1 Mix Dataset

In the *mix* setting we keep the clean hierarchical matrix but we corrupt the target correlation matrix C itself with unstructured noise, so that the model learns to denoise not only the finite sample but also a controlled amount of structural contamination. For each sample we draw a clean hierarchical correlation C_{clean} and an unstructured correlation matrix C_{u} obtained from an inverse-Wishart distribution. The population correlation is then defined as:

$$C = \mathcal{N}[(1 - w) C_{\text{clean}} + w C_{\text{u}}], \quad w \sim \mathcal{U}(w_{\text{min}}, w_{\text{max}}), \quad (\text{A.1})$$

where $\mathcal{N}[\cdot]$ denotes renormalization to a proper correlation matrix (unit diagonal) and the mixing weight w is drawn independently for each sample. As a consequence, within a single batch some matrices are almost purely hierarchical ($w \approx 0$) while others are dominated by unstructured noise ($w \approx 1$). The noisy input is then obtained, as in the base case, by estimating the correlation from T simulated observations of C . Here the target is the population matrix itself, so each dataset element is the pair $(\hat{C}, C)$. We can see an example of this in Figure A.1.

A.2 Overlapping Dataset

In the *overlap*-regime setting the goal is to reproduce markets whose dependency structure cannot be described by a single clean tree, but rather by two overlapping hierarchical regimes. To this end we sample two independent hierarchical correlation matrices, C_{primary} and $C_{\text{secondary}}$, and combine them into a single target through an overlap operation,

$$C = \Phi(C_{\text{primary}}, C_{\text{secondary}}), \quad (\text{A.2})$$

where $\Phi(\cdot, \cdot)$ superimposes the two structures so that assets do not fall into one clean grouping but into competing, overlapping ones. As before, a noisy realization $\hat{C}$ is estimated from T simulated observations, so each dataset element is the pair $(\hat{C}, C)$. We can see an example of this in Figure A.2. The model is therefore required to strip away both the finite-sample noise and any unstructured contamination, recovering the underlying dual-regime structure; this makes it the most demanding of the three denoising targets.

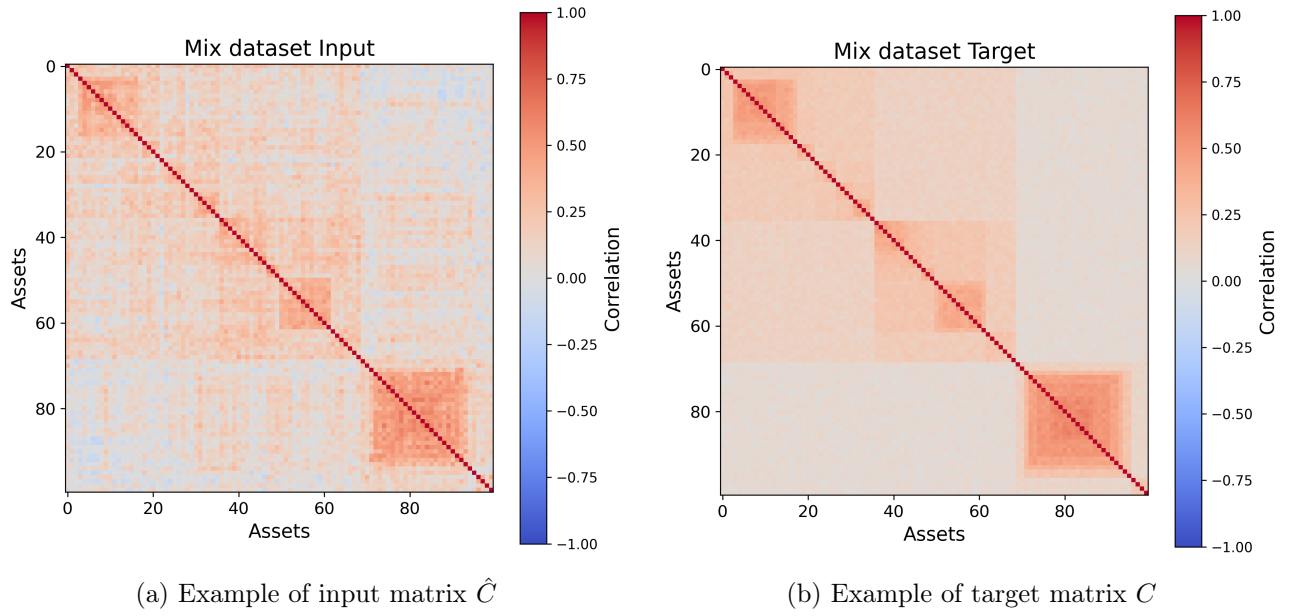


Figure A.1: Example of a pair of sample and target correlation matrices from the mix dataset.

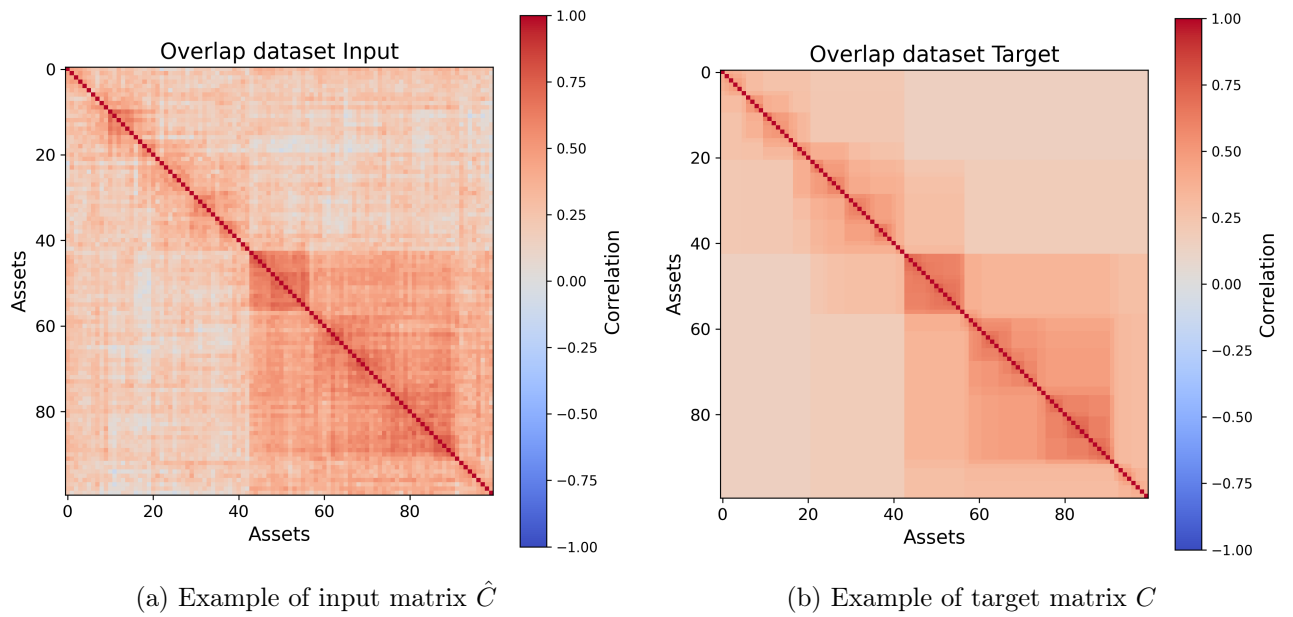


Figure A.2: Example of a pair of sample and target correlation matrices from the overlap dataset.

Appendix B

Returns and Drifted Weights

This appendix provides the mathematical derivation for the total return of a buy-and-hold portfolio, proving that the compounded return with drifted weights is mathematically equivalent to the weighted sum of the individual assets' cumulative returns based on their initial allocations. It is easy to prove that:

$$\prod_{t=0}^{T-1} (1 + \sum_i w_i^{(t)} r_i(t+1)) - 1 = \sum_i w_i^{(0)} (G_i(T) - 1) \quad (\text{B.1})$$

First, let us show that:

$$\frac{V(t+1)}{V(t)} = 1 + \sum_i w_i^{(t)} r_i(t+1) \quad (\text{B.2})$$

Proof:

$$\frac{V(t+1)}{V(t)} = \frac{\sum_i w_i^{(t)} V(t) (1 + r_i(t+1))}{V(t)} = \sum_i w_i^{(t)} (1 + r_i(t+1)) = 1 + \sum_i w_i^{(t)} r_i(t+1) \quad (\text{B.3})$$

Once we have this we can write:

$$\prod_{t=0}^{T-1} (1 + \sum_i w_i^{(t)} r_i(t+1)) - 1 = \prod_{t=0}^{T-1} \frac{V(t+1)}{V(t)} - 1 = \frac{V(T)}{V(0)} - 1 \quad (\text{B.4})$$

and this is equal to:

$$\frac{\sum_i w_i^{(0)} V(0) G_i(T)}{V(0)} - 1 = \sum_i w_i^{(0)} (G_i(T) - 1) \quad (\text{B.5})$$

This gave us the return of the portfolio keeping the weights fixed for the whole window.

Appendix C

Results on Synthetic Datasets

We report here the results of the experiments on the synthetic datasets described in Appendix A. In particular, in Table C.1 we report the results on the overlap dataset, while in Table C.2 we report the results on the mix dataset.

	MSE $\pm$ Std Error	95% CI
sample	0.0053 ± 0.0001	[0.0051, 0.0054]
GNN	0.0015 ± 0.0001	[0.0014, 0.0016]
BAHC	0.0017 ± 0.0001	[0.0016, 0.0019]

Table C.1: MSE on the *Overlap* dataset for the sample correlation, our reconstruction (*GNN*) and the BAHC benchmark. Values are reported as the mean over bootstrap resamples with the corresponding 95% confidence interval. Lower is better; the best value is in bold.

	MSE $\pm$ Std Error	95% CI
sample	0.0063 ± 0.0001	[0.0063, 0.0064]
GNN	0.0024 ± 0.0001	[0.0023, 0.0026]
BAHC	0.0037 ± 0.0001	[0.0035, 0.0040]

Table C.2: MSE on the *Mix* dataset for the sample correlation, our reconstruction (*GNN*) and the BAHC benchmark. Values are reported as the mean over bootstrap resamples with the corresponding 95% confidence interval. Lower is better; the best value is in bold.

Appendix D

Results on Real Datasets

This appendix provides further empirical results obtained from evaluating our estimators on real-world datasets.

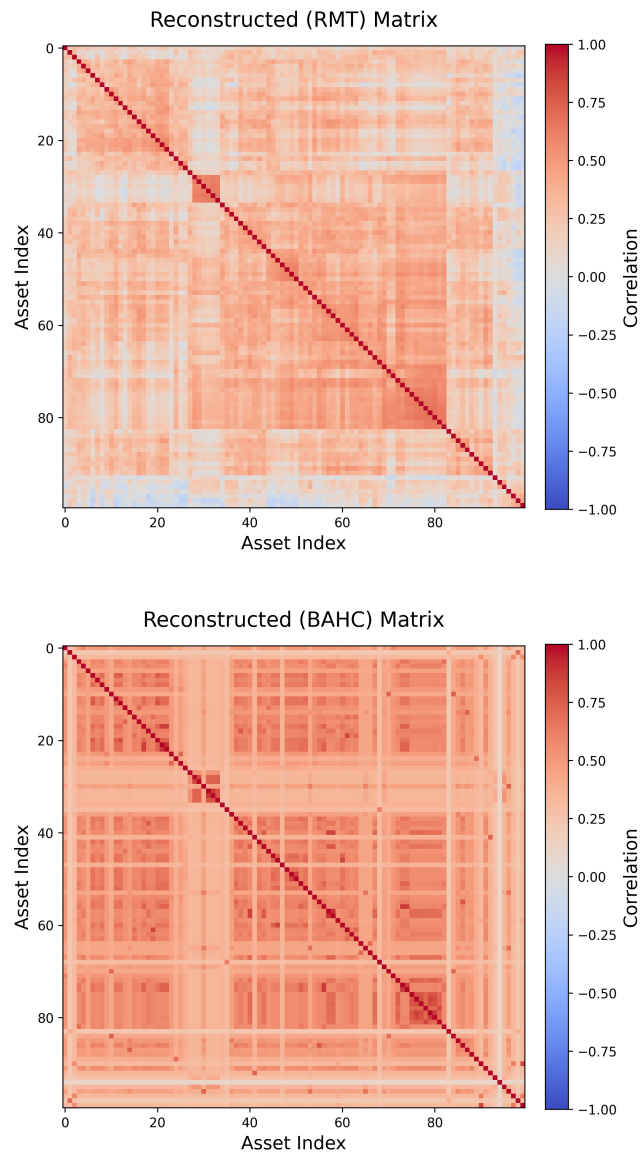


Figure D.1: Cleaned versions of the correlation matrix in Figure 4.10a using the RMT estimator (top) and the BAHC estimator (bottom).

Appendix E

Results on Long-Only Portfolios

In this section, we detail the formulation of the long-only portfolio optimisation problem and discuss the backtesting results achieved on real-world data. The weights of the long-only GMV portfolio are obtained by solving the following constrained optimisation problem:

$$\min_w w^\top \hat{\Sigma} w \quad \text{s.t.} \quad w^\top \mathbf{1} = 1, \quad w_i \geq 0, \forall i, \quad (\text{E.1})$$

where $\hat{\Sigma}$ is the estimated covariance matrix, w is the vector of portfolio weights, and $\mathbf{1}$ is a vector of ones. The first constraint ensures that the weights sum to one, while the second constraint enforces that all weights are non-negative, thus ensuring a long-only portfolio. This optimisation problem can be solved using quadratic programming techniques. We employed the OSQP solver built into `cvxpy` [12] to solve the optimisation problem for each covariance estimator. We report the results of the backtesting on real data in Table E.1.

Method	Ann. Return	Ann. Vol.	Sharpe	Cumulative	Turnover	N_{eff}
Sample	0.0707	0.1369	0.517	3.8768	2.8288	10.66
QIS	0.0712	0.1367	0.521	3.9417	2.6885	11.50
RMT	0.0701	0.1379	0.508	3.7811	2.8782	10.66
GNN	0.0784	0.1392	0.563	4.9038	2.7130	10.84
BAHC	0.0787	0.1374	0.573	4.9815	2.6419	10.99
EWP (<i>benchmark</i>)	0.0899	0.1939	0.464	5.2704	0.7571	99.62

Table E.1: Out-of-sample portfolio performance for each covariance estimator, with the equally weighted portfolio (EWP) reported separately as a naïve benchmark: annualised return, annualised volatility, Sharpe ratio, cumulative return, average turnover, and effective number of constituents N_{eff} . The best value in each column among the covariance estimators is shown in bold (highest return, Sharpe, and cumulative return; lowest volatility and turnover).

We can see that despite having a lower cumulative return than the EWP, the GNN-based and BAHC-based portfolios show the highest Sharpe ratios among the covariance estimators, with BAHC slightly ahead in this constrained setting. In conclusion, the results are not much different from the unconstrained case, and the GNN-based portfolio still shows a better risk-adjusted performance than the EWP. In Figure E.1 we report the cumulative returns of the GMV portfolio constructed using each covariance estimator, while in Figure E.2 we report the annualised volatility of the portfolio.

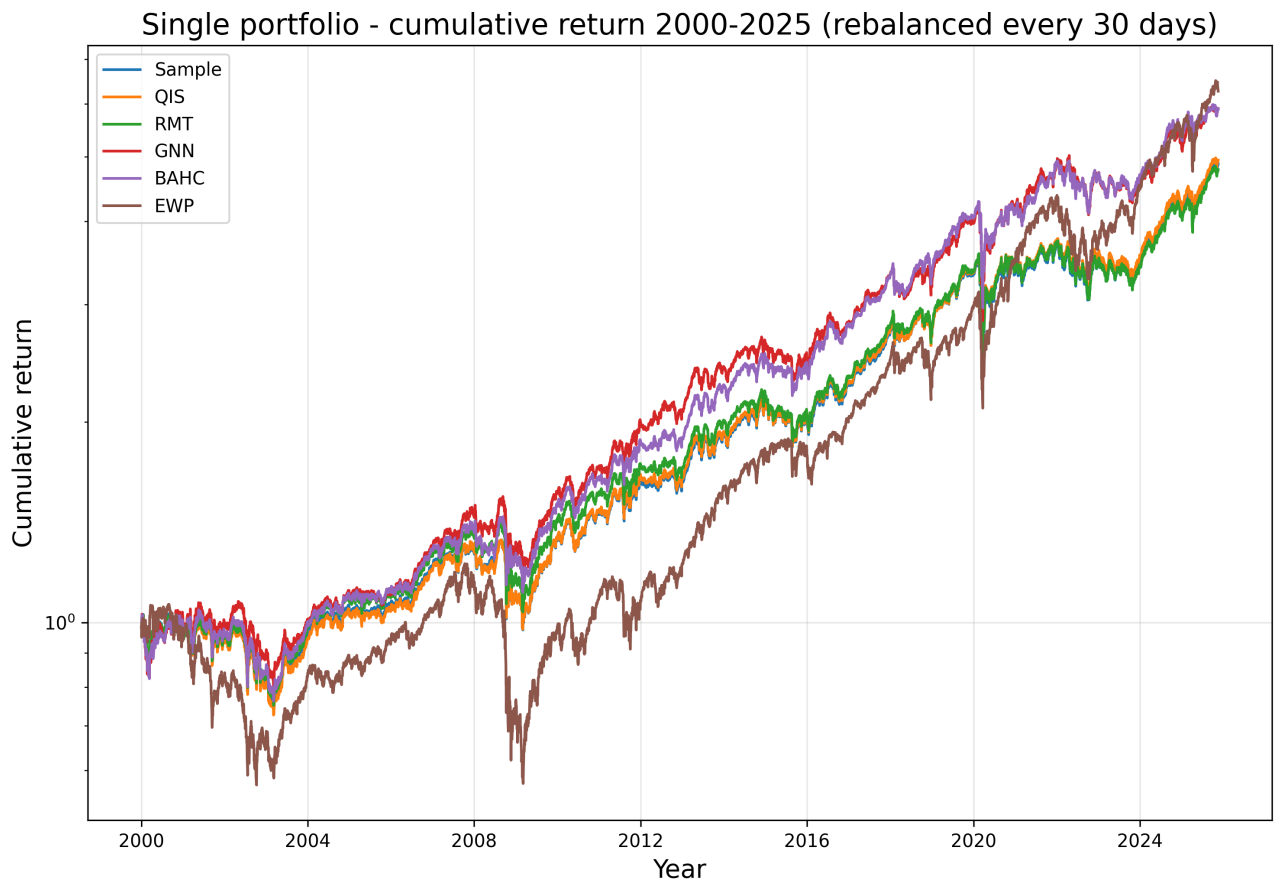


Figure E.1: Cumulative returns of the GMV portfolio constructed using each covariance estimator. The GNN-based portfolio shows the highest cumulative return over the evaluation period.

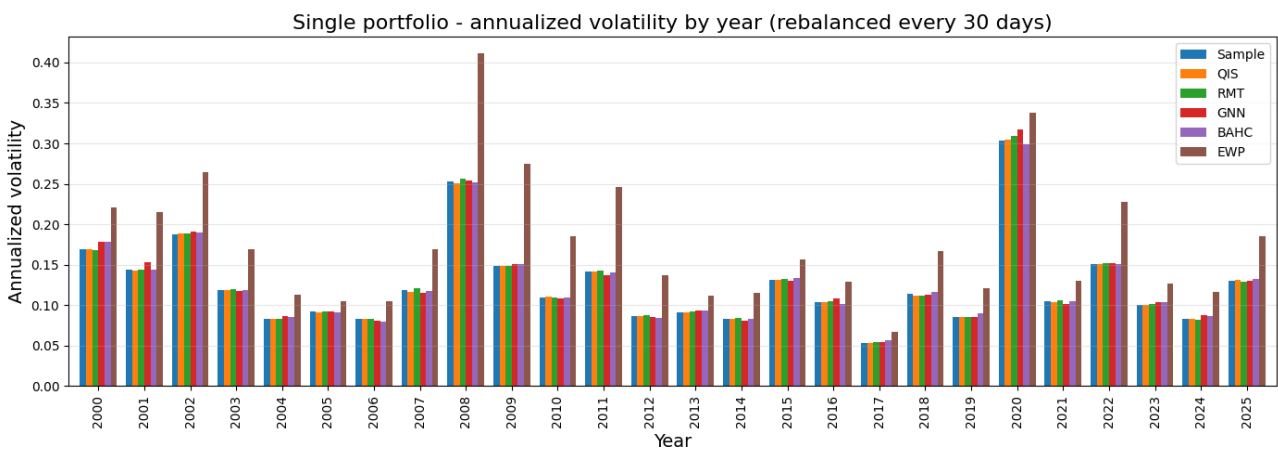


Figure E.2: Annualised volatility of the GMV portfolio constructed using each covariance estimator.

Appendix F

Residual Learning

The question raised in section 5.1 is whether the model is able to learn in a controlled setting the optimal linear shrinkage predicted by RMT. As we saw, this is not the case and this can be explained by the fact that the model is not rotationally invariant by construction and while message-passing schemes are good at learning local features, they are not able to learn global ones. Moreover, in the particular case of Inverse-Wishart matrices, we do not have structure to extract at all. What we can do is take inspiration from what is done in real pipelines: there is a first step of cleaning whose aim is to extract the structures from the data and then a second step of denoising that is applied to what is left, the residual. To denoise the residual we apply a NN called RIEnet [6]¹. It’s important to recall that, as explained in Section A.1, in this case the target is the population matrix itself, so also the target is noisy. As we saw, this is a more difficult task that is close to the real case.

In the Table F.1 we can see the results of the linear regression on the off-diagonal elements of the matrices in the mix dataset, and we can see that the model is able to learn a shrinkage factor that is very close to the theoretical one when the target is an inverse-Wishart matrix. Before moving to the

	$\beta \pm$ Std Error	95% CI
Theoretical shrinkage factor	0.6024	—
GNN	0.5827 ± 0.0023	[0.5781, 0.5875]
QIS	0.5836 ± 0.0020	[0.5797, 0.5878]

Table F.1: Shrinkage coefficient β estimated via linear regression on the off-diagonal elements. Standard deviation and 95% CI are obtained by bootstrapping over 500 matrices. The theoretical shrinkage factor is reported as a reference.

real data it is instructive to look at what the model is doing on the synthetic data when there is a clear structure in addition to the inverse-Wishart noise, and in the case in which there is no structure at all. In Figure F.1 we can see an example of input and target matrices from the mix dataset with a clear structure in addition to the inverse-Wishart noise, while in Figure F.3 we can see an example of input and target matrices from the mix dataset with no structure in addition to the inverse-Wishart noise. In Figure F.2 we can see the results of the residual learning on the input matrix in Figure F.1, while in Figure F.4 we can see the results of the residual learning on the input matrix in Figure F.3. It is interesting to note that in Figure F.2a we get a clear picture of the real underlying structure, while in F.4a where we have no structure at all, we get as output of the GNN the identity matrix. The two examples are produced with the same model trained on the mix dataset, and this tells us that the model is able to learn to extract the structure when it is present and to output the identity matrix when there is no structure at all.

¹The training of this model is more complex because, in order to prevent the RIEnet from doing all the work, we first need to train the GNN alone, then train the RIEnet with the GNN frozen, and finally train them together in the last few steps.

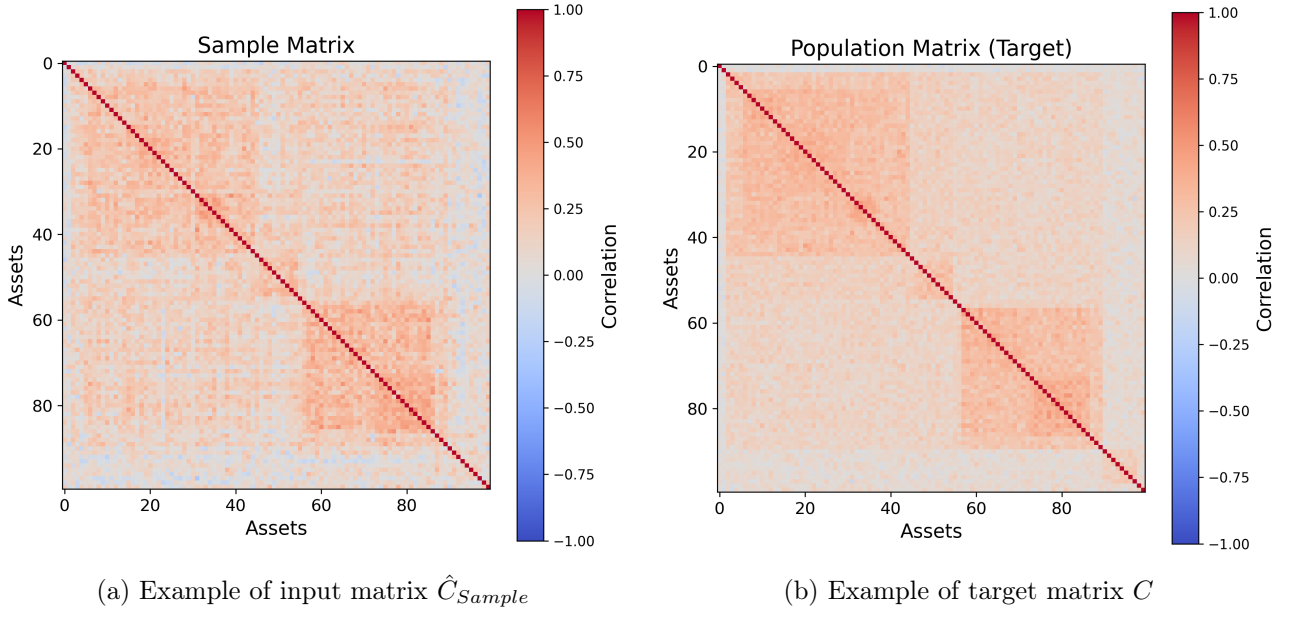


Figure F.1: Input and target matrices from mix dataset with a clear structure in addition to the inverse-Wishart noise.

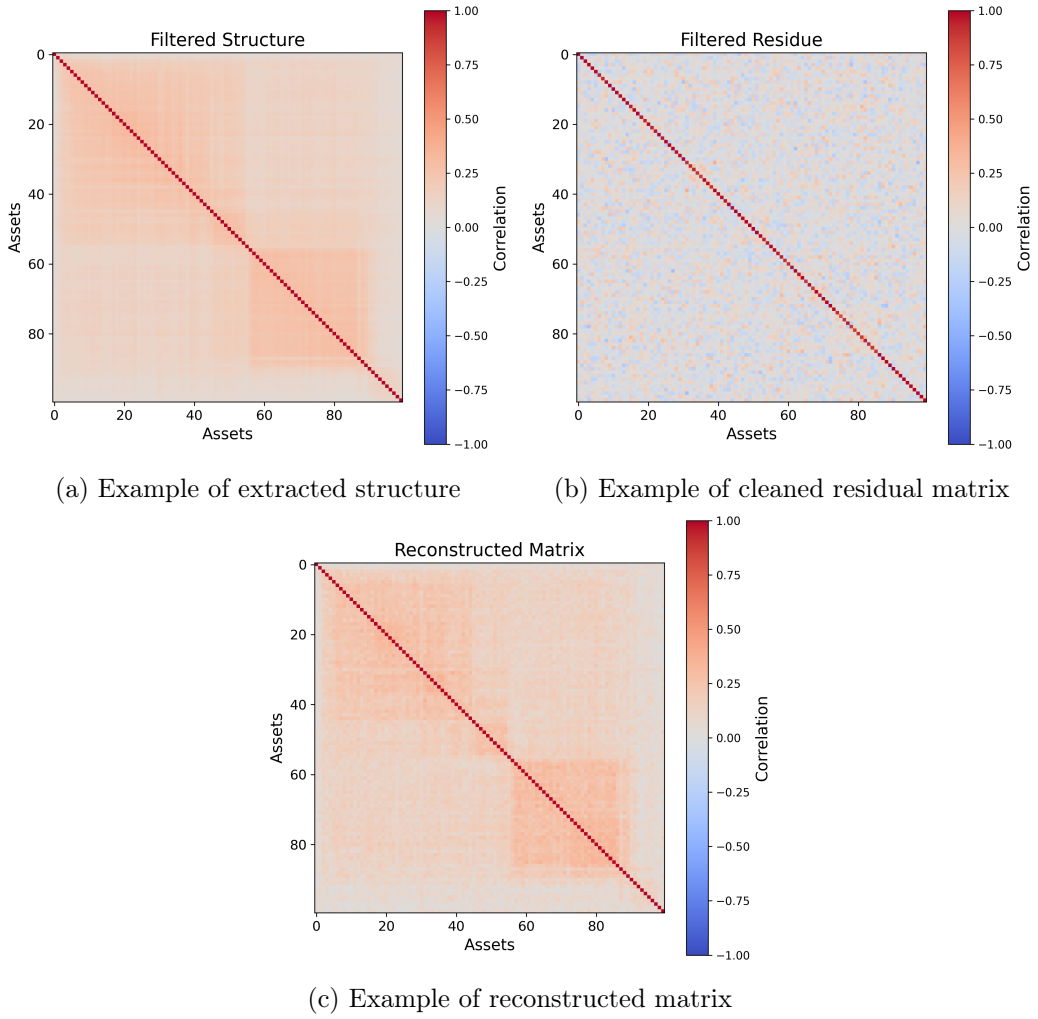


Figure F.2: Example of the results of the residual learning on the input matrix in Figure F.1. The first column shows the extracted structure, the second column shows the cleaned residual matrix and the third column shows the reconstructed matrix $\hat{C}$.

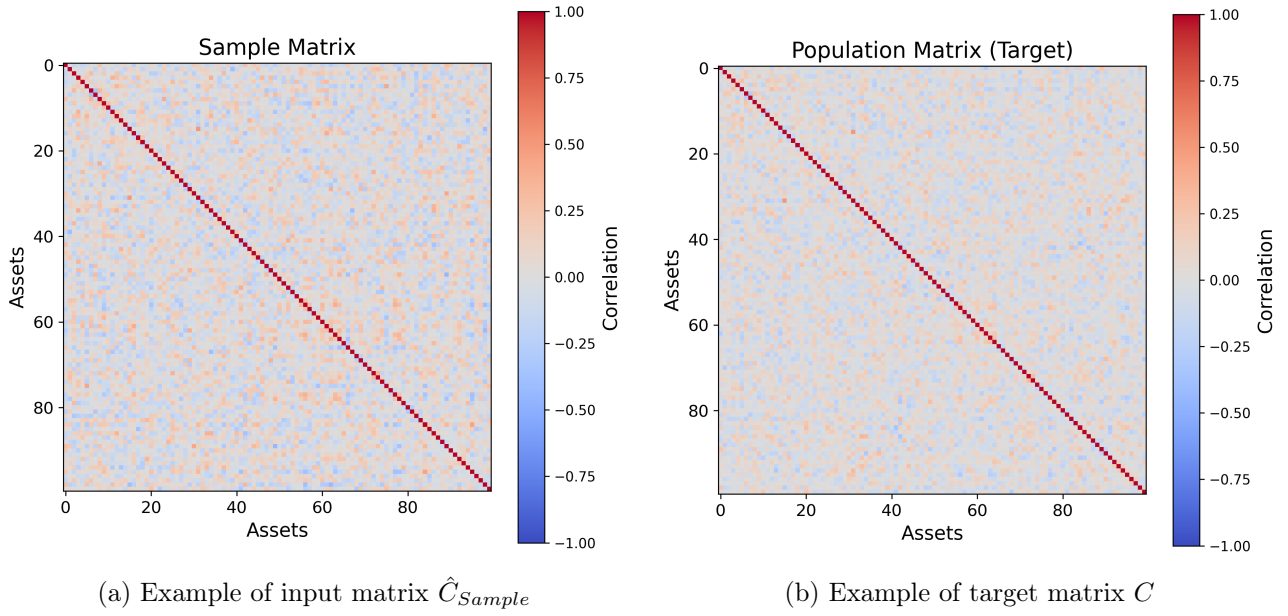


Figure F.3: Input and target matrices from mix dataset with no structure in addition to the inverse-Wishart noise.

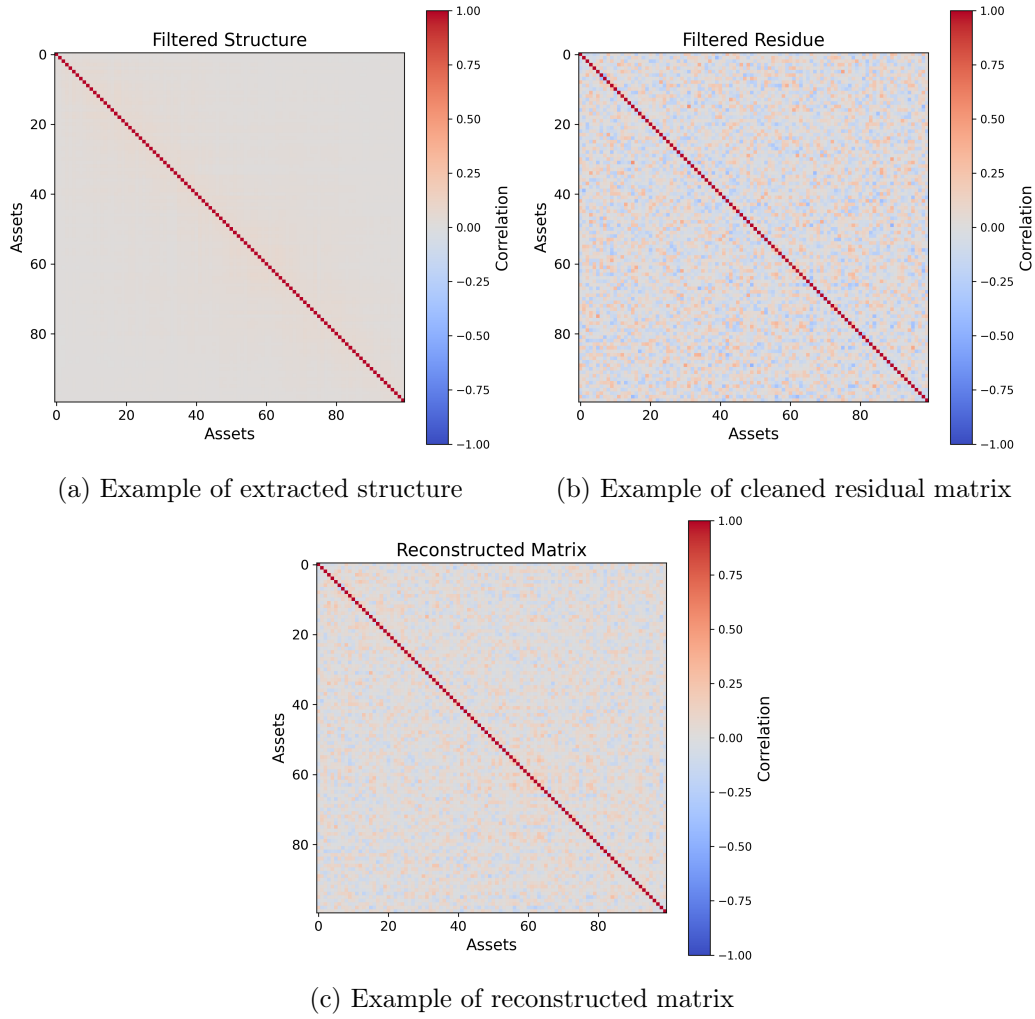


Figure F.4: Example of the results of the residual learning on the input matrix in Figure F.3. The first column shows the extracted structure, the second column shows the cleaned residual matrix and the third column shows the reconstructed matrix $\hat{C}$.

At this point we can move to the real data. In the following we will show the backtesting results of the model trained on real data with the residual learning approach.

Method	Ann. Return	Ann. Vol.	Sharpe	Cumulative	Turnover	N_{eff}
Sample	0.0466	0.1471	0.317	1.5171	15.1678	53.49
QIS	0.0663	0.1334	0.497	3.3969	9.6102	50.11
RMT	0.0728	0.1387	0.525	4.0954	10.7473	50.42
GNN (Residual)	0.0857	0.1348	0.636	6.2130	8.1798	46.29
BAHC	0.0686	0.1424	0.482	3.5191	8.7223	49.28
EWP (benchmark)	0.0871	0.1935	0.450	4.8248	0.1482	99.37

Table F.2: Out-of-sample portfolio performance for each covariance estimator, with the equally weighted portfolio (EWP) reported separately as a naïve benchmark: annualised return, annualised volatility, Sharpe ratio, cumulative return, average turnover, and effective number of constituents N_{eff} . The best value in each column *among the covariance estimators* is shown in bold (highest return, Sharpe, and cumulative return; lowest volatility and turnover).

In Table F.2 we can see the results of the backtesting on real data, and we can see that the model trained with the residual learning approach is able to outperform all the other methods in terms of annualized return, Sharpe ratio, cumulative return and turnover. It's interesting to notice that in this case the realized volatility of the model is lower compared to the normal GNN. Even if the realized volatility is not the lowest, we can see in Figure F.6 that the model is able to keep the volatility low and stable over time. In Figure F.5 we can see that the model is able to achieve the highest cumulative return over the evaluation period.

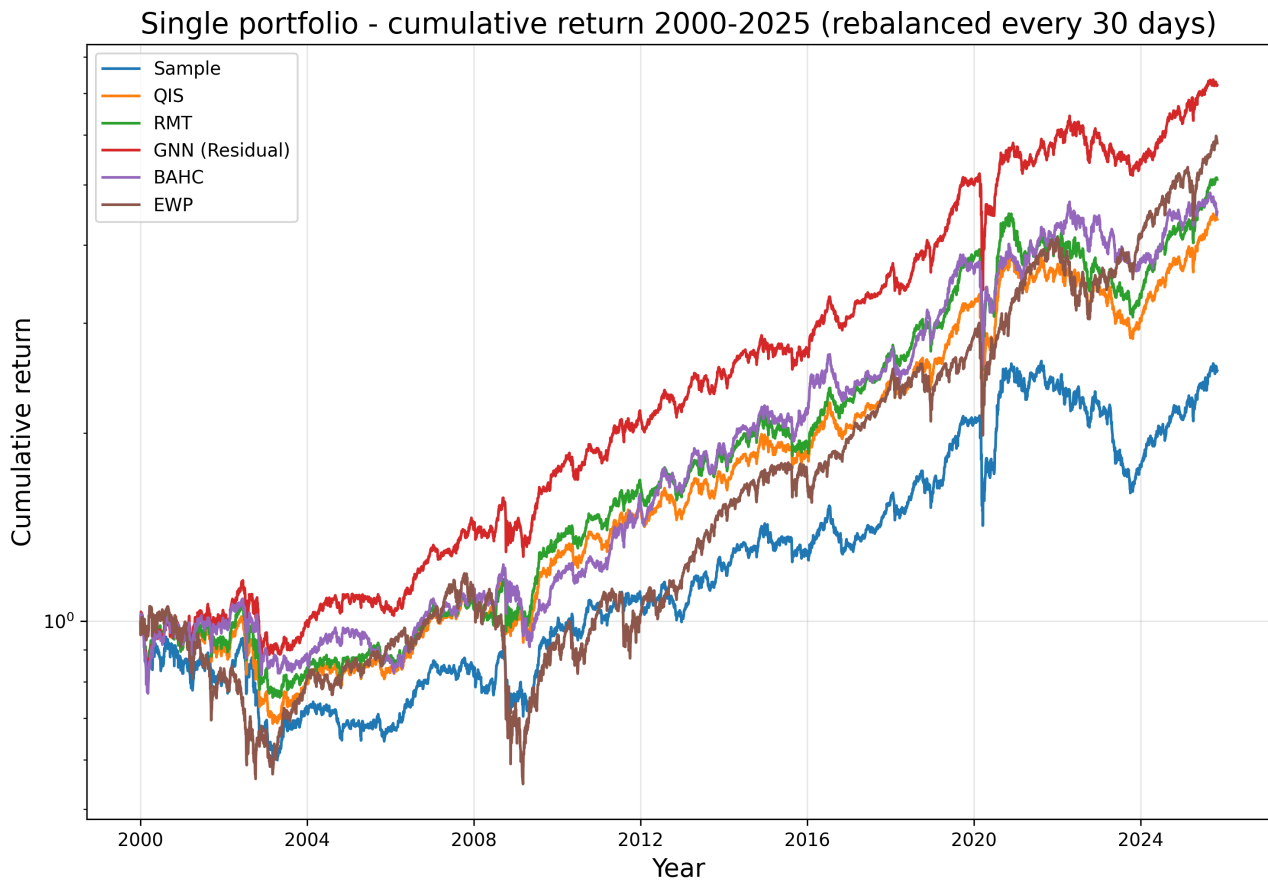


Figure F.5: Cumulative returns of the GMV portfolio constructed using each covariance estimator. The GNN-based portfolio shows the highest cumulative return over the evaluation period.

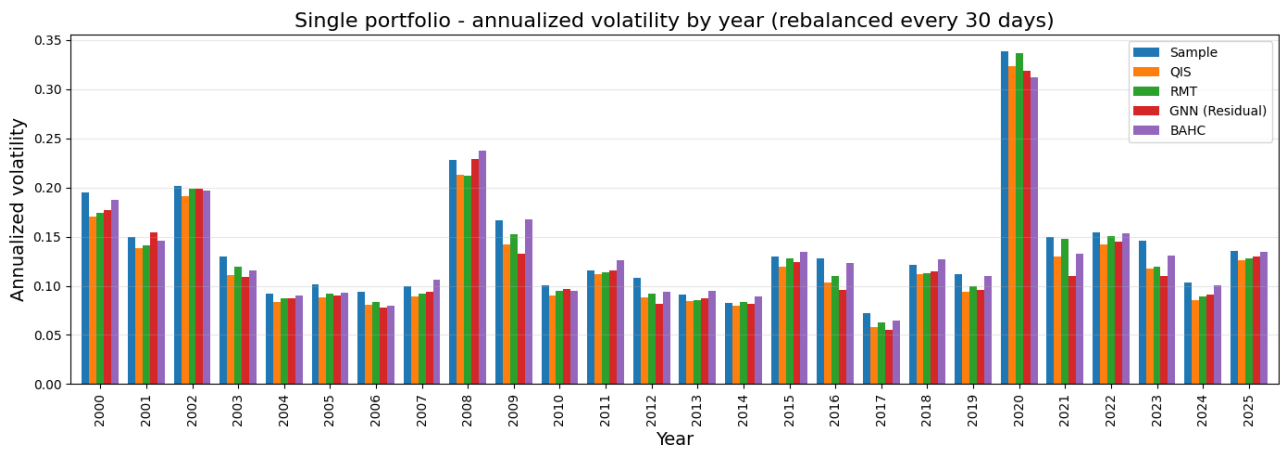


Figure F.6: Annualised volatility of the GMV portfolio constructed using each covariance estimator.

In conclusion, even if we are not training the model to learn the optimal GMV portfolio, we can see that the model both achieves a lower MSE and a better performance in terms of portfolio variance compared to the normal GNN.

Appendix G

KLD Loss

We already pointed out the fact that in order to achieve better performance in terms of portfolio variance we don't necessarily need to minimize the MSE between the predicted and the target correlation matrices. An alternative loss function for our architecture can be the Kullback-Leibler divergence (KLD) between the predicted and the target correlation matrices. Let Σ be the target correlation matrix of dimension $N \times N$, and $\hat{\Sigma}$ be the predicted matrix. We can associate these matrices with two zero-mean multivariate Gaussian distributions to isolate the covariance structure: the target distribution $P \sim \mathcal{N}(0, \Sigma)$ and the predicted distribution $Q \sim \mathcal{N}(0, \hat{\Sigma})$. The KLD measures the information lost when Q is used to approximate P . Since the means are zero, the divergence simplifies to:

$$D_{KL}(P \parallel Q) = \frac{1}{2} \left[\text{tr}(\hat{\Sigma}^{-1}\Sigma) - N + \log \left(\frac{\det \hat{\Sigma}}{\det \Sigma} \right) \right] \quad (\text{G.1})$$

This information-theoretic measure is fundamentally connected to the Stein Loss (or Stein's distance), a widely adopted asymmetric loss function in covariance matrix estimation [19], [18], which is defined as:

$$L(\Sigma, \hat{\Sigma}) = \text{tr}(\Sigma\hat{\Sigma}^{-1}) - \log \det(\Sigma\hat{\Sigma}^{-1}) - N \quad (\text{G.2})$$

By leveraging the cyclic property of the trace, $\text{tr}(\hat{\Sigma}^{-1}\Sigma) = \text{tr}(\Sigma\hat{\Sigma}^{-1})$, and the properties of the determinant, where $\log \left(\frac{\det \hat{\Sigma}}{\det \Sigma} \right) = -\log \det(\Sigma\hat{\Sigma}^{-1})$, we can rewrite the KLD to reveal a direct equivalence:

$$L(\Sigma, \hat{\Sigma}) = 2 \cdot D_{KL}(\mathcal{N}(0, \Sigma) \parallel \mathcal{N}(0, \hat{\Sigma})) \quad (\text{G.3})$$

By employing this formulation as our objective function, the architecture is trained to minimize the information loss between the true and predicted correlation structures. We tested this approach on the mix dataset and we found that the model trained with KLD loss is able to achieve a value of the optimal shrinkage factor which is near the one predicted by RMT for the inverse-Wishart case, as we can see in Table G.1.

	$\beta \pm$ Std Error	95% CI
Theoretical shrinkage factor	0.6024	—
GNN(KLD-loss)	0.5658 ± 0.0031	[0.5595, 0.5723]
QIS	0.5816 ± 0.0021	[0.5774, 0.5858]

Table G.1: Shrinkage coefficient β estimated via linear regression on the off-diagonal elements. Standard deviation and 95% confidence intervals are obtained by bootstrapping over 500 matrices. The theoretical shrinkage factor is reported as a reference.

We report in Figure G.1 an example of the results of the model trained with KLD loss on a sample from the mix dataset.

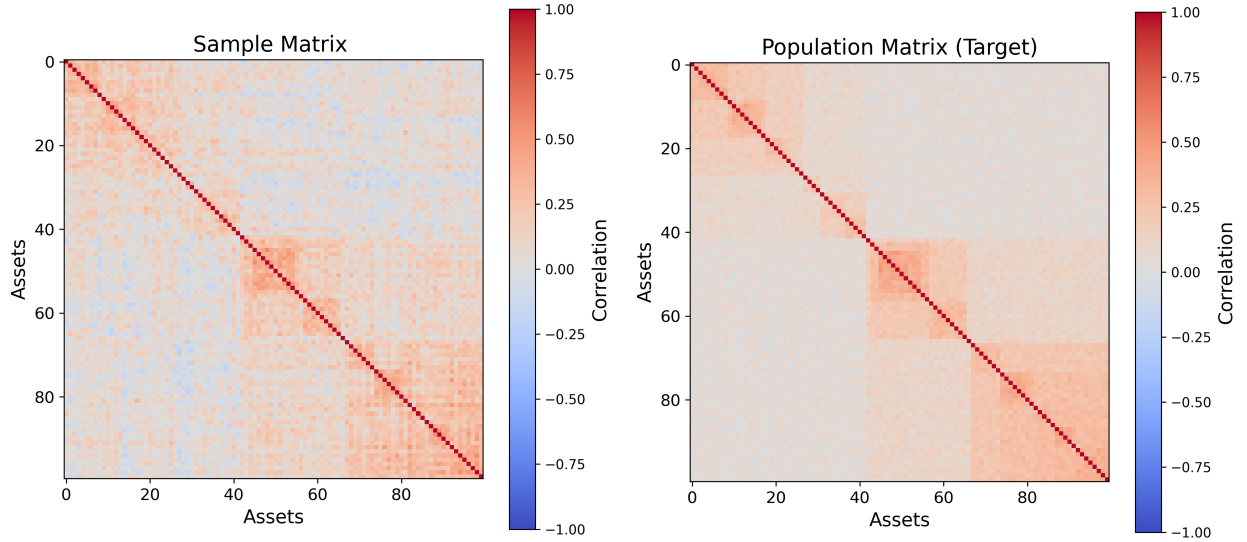
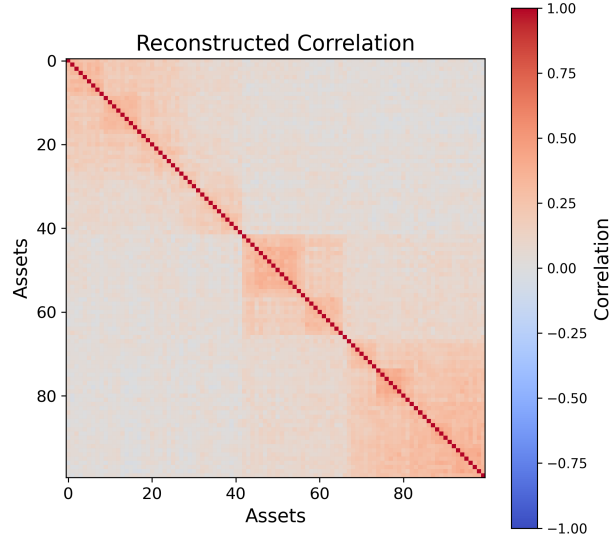
(a) Example of sample matrix $\hat{C}_{Sample}$ (b) Example of population matrix C (c) Example of reconstructed matrix $\hat{C}$

Figure G.1: Example of the results of the model trained with KLD loss on a sample from the mix dataset. The first column shows the sample matrix, the second column shows the population matrix and the third column shows the reconstructed matrix $\hat{C}$.

This seems like a promising approach; indeed now we have something in the loss that encourages the model to correct the eigenvalues towards the target ones. Despite this, when training on real data we need to use as target correlation matrices computed on a horizon of few days resulting in a singular target matrix, which makes the KLD loss not applicable. We leave the exploration of this approach to future work, and we believe that this problem can be solved by using a symmetrized version of the KLD loss.

Bibliography

- [1] J. Bergstra and Y. Bengio. “Random search for hyper-parameter optimization”. In: *J. Mach. Learn. Res.* 13.null (Feb. 2012), pp. 281–305. ISSN: 1532-4435.
- [2] C. Bongiorno and D. Challet. “Covariance matrix filtering with bootstrapped hierarchies”. In: *PLOS ONE* 16.1 (Jan. 2021), pp. 1–13. DOI: [10.1371/journal.pone.0245092](https://doi.org/10.1371/journal.pone.0245092). URL: <https://doi.org/10.1371/journal.pone.0245092>.
- [3] C. Bongiorno and D. Challet. “Non-linear shrinkage of the price return covariance matrix is far from optimal for portfolio optimization”. In: *Finance Research Letters* 52 (2023), p. 103383. ISSN: 1544-6123. DOI: <https://doi.org/10.1016/j.frl.2022.103383>. URL: <https://www.sciencedirect.com/science/article/pii/S1544612322005608>.
- [4] C. Bongiorno and D. Challet. “Reactive Global Minimum Variance Portfolios with k -BAHC covariance cleaning”. In: *European Journal of Finance* 28.13-15 (Aug. 2021), pp. 1344–1360. DOI: [10.1080/1351847X.2021.1963301](https://doi.org/10.1080/1351847X.2021.1963301). URL: <https://centralesupelec.hal.science/hal-02612262>.
- [5] C. Bongiorno and L. Lamrani. “Quantifying the information lost in optimal covariance matrix cleaning”. In: *Physica A: Statistical Mechanics and its Applications* 657 (2025), p. 130225. ISSN: 0378-4371. DOI: <https://doi.org/10.1016/j.physa.2024.130225>. URL: <https://www.sciencedirect.com/science/article/pii/S0378437124007349>.
- [6] C. Bongiorno, E. Manolakis, and R. N. Mantegna. “End-to-end large portfolio optimization for variance minimization with neural networks through covariance cleaning”. In: *The Journal of Finance and Data Science* 12 (2026), p. 100179. ISSN: 2405-9188. DOI: <https://doi.org/10.1016/j.jfds.2026.100179>. URL: <https://www.sciencedirect.com/science/article/pii/S2405918826000048>.
- [7] J.-P. Bouchaud and M. Potters. *Theory of Financial Risks; Theory of Financial Risks and Derivative Pricing. From Statistical Physics to Risk Management*. Cambridge University Press, 2003 (2nd Edition), 2000. URL: <https://hal.science/hal-00121107>.
- [8] J.-P. Bouchaud and M. Potters. “Financial applications of random matrix theory: a short review”. In: *The Oxford Handbook of Random Matrix Theory*. Ed. by G. Akemann, J. Baik, and P. Di Francesco. Oxford University Press, Sept. 2015. ISBN: 9780198744191. DOI: [10.1093/oxfordhb/9780198744191.013.40](https://doi.org/10.1093/oxfordhb/9780198744191.013.40). eprint: https://academic.oup.com/book/0/chapter/365887049/chapter-ag-pdf/45323208/book_43656_section_365887049.ag.pdf. URL: <https://doi.org/10.1093/oxfordhb/9780198744191.013.40>.
- [9] J. Bun, J.-P. Bouchaud, and M. Potters. *My Beautiful Laundrette: Cleaning Correlation Matrices for Portfolio Optimization*. May 2016. DOI: [10.13140/RG.2.1.2782.2961](https://doi.org/10.13140/RG.2.1.2782.2961).
- [10] J. Bun, J.-P. Bouchaud, and M. Potters. “Cleaning large correlation matrices: Tools from Random Matrix Theory”. In: *Physics Reports* 666 (Jan. 2017), pp. 1–109. ISSN: 0370-1573. DOI: [10.1016/j.physrep.2016.10.005](https://doi.org/10.1016/j.physrep.2016.10.005). URL: <http://dx.doi.org/10.1016/j.physrep.2016.10.005>.
- [11] C. Daniel. “One-at-a-Time Plans”. In: *Journal of the American Statistical Association* 68.342 (1973), pp. 353–360. ISSN: 01621459, 1537274X. URL: <http://www.jstor.org/stable/2284076> (visited on 07/10/2026).

- [12] S. Diamond and S. Boyd. “CVXPY: A Python-embedded modeling language for convex optimization”. In: *Journal of Machine Learning Research* 17.83 (2016), pp. 1–5.
- [13] S. Huang, L. Cao, R. Sun, T. Ma, and S. Liu. “Enhancing portfolio optimization: A two-stage approach with deep learning and portfolio optimization”. In: *Mathematics* 12.21 (2024), p. 3376.
- [14] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: [1412.6980](https://arxiv.org/abs/1412.6980) [cs.LG]. URL: <https://arxiv.org/abs/1412.6980>.
- [15] L. Laloux, P. Cizeau, J.-P. Bouchaud, and M. Potters. “Noise Dressing of Financial Correlation Matrices”. In: *Phys. Rev. Lett.* 83 (7 Aug. 1999), pp. 1467–1470. DOI: [10.1103/PhysRevLett.83.1467](https://doi.org/10.1103/PhysRevLett.83.1467). URL: <https://link.aps.org/doi/10.1103/PhysRevLett.83.1467>.
- [16] O. Ledoit and S. Péché. “Eigenvectors of some large sample covariance matrix ensembles”. In: *Probability Theory and Related Fields* 151.1 (2011), pp. 233–264. DOI: [10.1007/s00440-010-0298-3](https://doi.org/10.1007/s00440-010-0298-3). URL: <https://doi.org/10.1007/s00440-010-0298-3>.
- [17] O. Ledoit and M. Wolf. “A well-conditioned estimator for large-dimensional covariance matrices”. In: *Journal of Multivariate Analysis* 88.2 (2004), pp. 365–411. ISSN: 0047-259X. DOI: [https://doi.org/10.1016/S0047-259X\(03\)00096-4](https://doi.org/10.1016/S0047-259X(03)00096-4). URL: <https://www.sciencedirect.com/science/article/pii/S0047259X03000964>.
- [18] O. Ledoit and M. Wolf. “Nonlinear Shrinkage of the Covariance Matrix for Portfolio Selection: Markowitz Meets Goldilocks”. In: *The Review of Financial Studies* 30.12 (Dec. 2017), pp. 4349–4388. ISSN: 0893-9454. DOI: [10.1093/rfs/hhx052](https://doi.org/10.1093/rfs/hhx052). eprint: <https://academic.oup.com/rfs/article-pdf/30/12/4349/24433779/hhx052.pdf>. URL: <https://doi.org/10.1093/rfs/hhx052>.
- [19] O. Ledoit and M. Wolf. “Quadratic shrinkage for large covariance matrices”. In: *Bernoulli* 28.3 (2022), pp. 1519–1547. DOI: [10.3150/20-BEJ1315](https://doi.org/10.3150/20-BEJ1315). URL: <https://doi.org/10.3150/20-BEJ1315>.
- [20] Q. Li, Z. Han, and X.-M. Wu. *Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning*. 2018. arXiv: [1801.07606](https://arxiv.org/abs/1801.07606) [cs.LG]. URL: <https://arxiv.org/abs/1801.07606>.
- [21] R. N. Mantegna. “Hierarchical structure in financial markets”. In: *The European Physical Journal B - Condensed Matter and Complex Systems* 11.1 (1999), pp. 193–197. ISSN: 1434-6036. DOI: [10.1007/s100510050929](https://doi.org/10.1007/s100510050929). URL: <https://doi.org/10.1007/s100510050929>.
- [22] H. Markowitz. “PORTFOLIO SELECTION”. In: *The Journal of Finance* 7.1 (1952), pp. 77–91. DOI: <https://doi.org/10.1111/j.1540-6261.1952.tb01525.x>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1540-6261.1952.tb01525.x>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6261.1952.tb01525.x>.
- [23] D. P. Palomar. *Portfolio Optimization: Theory and Application*. Cambridge University Press, 2025.
- [24] P. Ramachandran, B. Zoph, and Q. V. Le. “Searching for Activation Functions”. In: *ArXiv abs/1710.05941* (2018). URL: <https://api.semanticscholar.org/CorpusID:10919244>.
- [25] P. Reis, A. P. Serra, and J. Gama. “A deep learning framework for medium-term covariance forecasting in multi-asset portfolios”. In: *arXiv preprint arXiv:2503.01581* (2025).
- [26] A. Saltelli and P. Annoni. “How to avoid a perfunctory sensitivity analysis”. In: *Environmental Modelling & Software* 25 (Dec. 2010), pp. 1508–1517. DOI: [10.1016/j.envsoft.2010.04.012](https://doi.org/10.1016/j.envsoft.2010.04.012).
- [27] M. Tumminello, F. Lillo, and R. N. Mantegna. “Hierarchically nested factor model from multivariate data”. In: *Europhysics Letters (EPL)* 78.3 (Apr. 2007), p. 30006. ISSN: 1286-4854. DOI: [10.1209/0295-5075/78/30006](https://doi.org/10.1209/0295-5075/78/30006). URL: <http://dx.doi.org/10.1209/0295-5075/78/30006>.
- [28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. “Attention Is All You Need”. In: *CoRR abs/1706.03762* (2017). arXiv: [1706.03762](https://arxiv.org/abs/1706.03762). URL: <https://arxiv.org/abs/1706.03762>.

- [29] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. “A Comprehensive Survey on Graph Neural Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (Jan. 2021), pp. 4–24. ISSN: 2162-2388. DOI: [10.1109/tnnls.2020.2978386](https://doi.org/10.1109/TNNLS.2020.2978386). URL: <http://dx.doi.org/10.1109/TNNLS.2020.2978386>.
- [30] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. *How Powerful are Graph Neural Networks?* 2019. arXiv: [1810.00826](https://arxiv.org/abs/1810.00826) [cs.LG]. URL: <https://arxiv.org/abs/1810.00826>.