universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Spectral–Temporal Graph Neural Networks for Demand Forecasting on Bipartite Supply–Customer Graphs

verfasst von | submitted by

Mahdi Mohaddes

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2026

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 821
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Mathematik
Betreut von | Supervisor:	Dipl.-Ing. Dr.techn. Lukas Exl Privatdoz.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Dr. Lukas Exl (Priv.-Doz.), for his invaluable guidance, continuous support, and constructive feedback throughout the course of this research. His expertise and encouragement were instrumental in shaping this thesis into its final form.

I would also like to extend my appreciation to Coveris GmbH for generously providing access to their data, which made this research possible. Their openness to academic collaboration and willingness to support scientific inquiry are greatly valued.

Finally, I am deeply grateful to family for their unwavering support, patience, and encouragement throughout my studies. Their belief in me has been a constant source of motivation.

Large language model tools, including ChatGPT, Claude, and Gemini, were used during the preparation of this thesis for editorial support, LaTeX assistance, and occasional support with Python syntax and debugging. Grammarly was additionally used for grammar and language checking. All generated content was reviewed and verified by me. I take full responsibility for the content of this thesis.

Abstract

This thesis investigates whether explicit customer–product graph structure improves monthly industrial demand forecasting. The study adapts a TGGC-style spectral–temporal graph neural network to two graph formulations: a fixed customer–Site–Product bipartite graph and a latent Site–Product graph learned from the time series. The architecture is adapted with several spectral filter bases — polynomial (Gegenbauer, Jacobi) and rational (ARMA, Cayley) — applied under each graph formulation.

The models are evaluated on a real-world company dataset using a 12-month lookback window to predict a 3-month horizon. Performance is measured with MAE, RMSE, and WAPE, with WAPE used as the main comparison metric because it relates total absolute error to total absolute sales volume. The empirical results show that simple temporal persistence is the strongest forecasting signal in the dataset. A three-month moving average achieves the best Site–Product WAPE and outperforms all tested neural architectures. Nevertheless, the experiments show that graph construction and spectral filter choice affect forecasting behaviour. The fixed bipartite graph performs close to the learned latent graph on the common Site–Product node set, suggesting that interpretable customer–product structure can be incorporated with limited loss relative to the learned graph, although it does not improve over the simple baseline. Among the graph neural models, an ARMA filter performs best for the fixed bipartite graph (WAPE 35.73 $\pm$ 3.30%), while a Cayley filter performs best for the latent Site–Product graph (WAPE 34.75 $\pm$ 3.16%). The fixed bipartite model is therefore slightly worse in WAPE than the latent Site–Product model, but the gap is small. This suggests that explicit and interpretable customer–product relations can be integrated into spectral–temporal forecasting with only a limited loss of predictive accuracy relative to the learned latent graph.

Kurzfassung

Diese Masterarbeit untersucht, ob eine explizite Kunden–Produkt-Graphstruktur die monatliche industrielle Nachfrageprognose verbessern kann. Dafür wird ein TGGC-ähnliches spektral–temporales Graph Neural Network auf zwei Graphformulierungen angewendet: einen festen bipartiten Kunden–Site–Produkt-Graphen sowie einen latenten Site–Produkt-Graphen, dessen Struktur aus den Zeitreihen gelernt wird. Die Basisarchitektur wird zudem verwendet, um verschiedene spektrale Filterbasen zu vergleichen, darunter polynomiale Filter wie Gegenbauer- und Jacobi-Filter sowie rationale Filter wie ARMA- und Cayley-Filter.

Die Modelle werden auf einem realen Unternehmensdatensatz evaluiert. Dabei wird ein Lookback-Fenster von zwölf Monaten verwendet, um einen Prognosehorizont von drei Monaten vorherzusagen. Die Modellgüte wird anhand von MAE, RMSE und WAPE bewertet. WAPE dient als Hauptvergleichsmetrik, da diese Kennzahl den gesamten absoluten Prognosefehler ins Verhältnis zum gesamten absoluten Verkaufsvolumen setzt und damit für Verkaufsdaten mit stark unterschiedlichen Volumina besonders geeignet ist.

Die empirischen Ergebnisse zeigen, dass einfache zeitliche Persistenz das stärkste Prognosesignal im Datensatz darstellt. Ein gleitender Drei-Monats-Durchschnitt erreicht den besten WAPE-Wert auf den Site–Produkt-Knoten und übertrifft alle getesteten neuronalen Architekturen. Dennoch zeigen die Experimente, dass sowohl die Graphkonstruktion als auch die Wahl des spektralen Filters das Prognoseverhalten beeinflussen. Der feste bipartite Graph erzielt auf der gemeinsamen Site–Produkt-Knotenmenge eine ähnliche Leistung wie der gelernte latente Graph. Dies deutet darauf hin, dass eine interpretierbare Kunden–Produkt-Struktur mit nur begrenztem Genauigkeitsverlust gegenüber einem gelernten Graphen in die spektral–temporale Prognose integriert werden kann, auch wenn sie den einfachen Basisansatz nicht verbessert.

Unter den Graph-Neural-Network-Modellen erzielt ein ARMA-Filter die beste Leistung für den festen bipartiten Graphen mit einem WAPE von 35,73 ± 3,30%. Für den latenten Site–Produkt-Graphen erzielt ein Cayley-Filter die beste Leistung mit einem WAPE von 34,75 ± 3,16%. Das feste bipartite Modell ist damit im WAPE etwas schwächer als das latente Site–Produkt-Modell, jedoch ist der Unterschied gering. Insgesamt zeigen die Ergebnisse, dass explizite und interpretierbare Kunden–Produkt-Beziehungen in spektral–temporale Prognosemodelle integriert werden können, ohne einen großen Genauigkeitsverlust gegenüber einer gelernten latenten Graphstruktur zu verursachen.

Contents

List of Tables

List of Figures

1. Introduction

1.1. Motivation

Forecasting is a central problem in operational decision-making. In many business settings, future demand is not observed directly, but decisions must still be made before the future is realized. Production planning, procurement, stock allocation, capacity planning, and sales budgeting all depend on some estimate of future demand. Time series forecasting provides a mathematical framework for this problem by using historical observations to predict future values [1, 2].

In industrial sales data, forecasting is especially challenging because the data are not only temporal but also relational. A sale is not just a number observed at a month. It is connected to a customer, a site, and a product category. These relationships contain information. Customers may repeatedly buy certain products, products may share similar demand patterns, and sites may behave differently because of operational or market conditions. Therefore, a forecasting model that ignores the structure between customers and products may lose useful information.

The aim of this thesis is to study whether such relational information can be represented through a graph and used for sales forecasting. More precisely, the thesis investigates a TGGC-style spectral–temporal graph neural network for monthly Net Sales forecasting. The main proposed construction is a fixed bipartite graph between customer groups and Site–Product nodes. This graph is compared with a Site–Product-only graph where the graph structure is learned from the time series. In this way, the thesis studies not only forecasting accuracy but also the role of the graph representation and the spectral graph filter.

This work is motivated by two observations. First, time series forecasting has developed from classical statistical models to a broad set of machine learning and deep learning architectures, including RNNs, CNNs, Transformers, and GNNs [2, 3, 4]. Second, recent surveys emphasize that no single architecture dominates all time series forecasting problems; instead, model performance depends strongly on the data characteristics, such as temporal dependence, distribution shift, and channel dependency [2]. The sales data considered in this thesis have exactly this type of structure: temporal dynamics combined with relationships between many related entities.

1.2. Time series data

A time series is a sequence of observations ordered in time. Formally, a univariate time series can be written as

$$\{y_t\}_{t=1}^{T},$$

where $y_t \in \mathbb{R}$ is the value observed at time t. The order of the observations is essential. In contrast to an ordinary independent sample, the value at time t may depend on previous values $y_{t-1}, y_{t-2}, \ldots$. This temporal dependence is one of the defining features of time series data [5, 1].

Time series often contain several components. A trend describes a long-term increase or decrease. Seasonality describes patterns that repeat at a fixed period. Cycles represent longer fluctuations that may not have a strictly fixed period. Noise and irregular values represent random or unusual deviations from the usual pattern. Non-stationarity occurs when statistical properties such as the mean or variance change over time [6, 2]. In practical forecasting, these components are often mixed together. This makes the problem mathematically interesting and practically difficult.

In this thesis, the time index is monthly. The observed variable is the actual Net Sales value. After aggregation, each node has a monthly time series. Depending on the graph construction, a node can be either a customer group or a Site–Product combination. Thus, the data are not treated only as one isolated sequence, but as a collection of related sequences.

1.3. Time series forecasting

Time series forecasting is the task of predicting future values using historical observations. Given a lookback window

$$(y_{t-W+1}, \ldots, y_t),$$

the goal is to predict one or more future values

$$(y_{t+1}, \ldots, y_{t+H}),$$

where W is the input window length and H is the forecast horizon.

In a one-step forecast, $H = 1$. In a multi-step forecast, $H > 1$. Multi-step forecasting is more difficult because uncertainty usually increases as the prediction moves farther away from the last observed value. A model may produce accurate predictions for the first forecast step but progressively larger errors for the second and third steps. For this reason, the results in this thesis are also analyzed by forecast horizon.

The literature distinguishes between short-term and long-term forecasting. Short-term forecasting concerns the near future and is often important for operational decisions. Long-term forecasting concerns more distant horizons and is more relevant for strategic planning [2]. The experiments in this thesis are best interpreted as short-term to medium-term monthly forecasting: the model predicts up to three months ahead using a one-year input window.

1.4. Univariate and multivariate forecasting

In univariate time series forecasting, the observation at each time step is a scalar,

$$y_t \in \mathbb{R}.$$

The forecasting problem can be written as

$$\widehat{y}_{t+1:t+H} = f(y_{t-W+1:t}),$$

where W is the input window length and H is the forecast horizon.

In multivariate time series forecasting, several variables are observed at each time step. The observation at time t is therefore a vector

$$X_t = (x_{t,1}, \ldots, x_{t,N})^\top \in \mathbb{R}^N.$$

The forecasting problem becomes

$$\widehat{X}_{t+1:t+H} = f(X_{t-W+1:t}),$$

where

$$X_{t-W+1:t} = (X_{t-W+1}, \ldots, X_t) \in \mathbb{R}^{W \times N}$$

and

$$\widehat{X}_{t+1:t+H} \in \mathbb{R}^{H \times N}.$$

Thus, the main difference is that the univariate case predicts a scalar-valued sequence, while the multivariate case predicts a vector-valued sequence. This setting is more suitable for the sales data in this thesis because many customer and product series are observed at the same time. Their interactions may contain useful information.

Multivariate forecasting also introduces an additional difficulty: the model must learn temporal relationships and cross-sectional relationships. The survey by Kim et al. emphasizes channel dependency as one of the important open challenges in modern time series forecasting [2]. In this thesis, the cross-sectional relationships are represented by a graph. The nodes are sales series, and the edges encode either fixed customer–product relations or learned similarity among Site–Product nodes.

1.5. Graph-based view of multivariate sales forecasting

A graph is a natural way to describe relationships between multiple time series. Let

$$G = (V, E)$$

be a graph with node set V and edge set E. A graph signal is a function on the nodes of the graph. At each month t, the sales values form a graph signal

$$X_t \in \mathbb{R}^{|V|}.$$

If two nodes are connected, the model exchanges information between them via graph convolution..

For the main model of this thesis, the graph is bipartite. One side contains customer group nodes and the other side contains Site–Product nodes. The adjacency matrix has the form

$$A = \begin{pmatrix} 0 & B \\ B^\top & 0 \end{pmatrix},$$

where B contains the customer–Site–Product interaction weights. This graph is fixed because it is constructed from observed historical transaction relations. It therefore has a direct business interpretation.

The second graph setting uses only Site–Product nodes. In that case, the graph is learned from the time series by the latent-correlation layer of the TGGC-style model. This setting is included because the fixed bipartite graph may be sparse on the customer side. Comparing the two graph constructions allows the thesis to ask whether explicit customer–product structure helps, weakens, or approximately matches the forecasting of Site–Product nodes.

1.6. Evaluation of forecasting models

Forecasting models must be evaluated on data that were not used for training. Since time series are ordered, the split must respect chronology. In this thesis, the data are divided into training, validation, and test periods in temporal order. This avoids using future information when training the model.

Several metrics are used because no single metric captures all aspects of model quality. The Mean Absolute Error is

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \widehat{y}_i|.$$

It measures the average absolute error in the original scale. The Root Mean Squared Error is

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2}.$$

RMSE gives more weight to large errors because the errors are squared before averaging. These two metrics are among the most common deterministic forecasting metrics [2, 6].

The main relative metric used in the thesis is the Weighted Absolute Percentage Error:

$$\mathrm{WAPE} = \frac{\sum_{i=1}^{n} |y_i - \widehat{y}_i|}{\sum_{i=1}^{n} |y_i|}.$$

WAPE is useful in this application because sales volumes differ strongly across nodes. It measures total absolute error relative to total absolute actual sales. This makes it more stable than node-wise percentage errors when some observations are close to zero.

The literature emphasizes that different evaluation metrics highlight different aspects of forecasting performance, and that relative metrics are useful when scale matters [2].

In the bipartite experiment, metrics are reported in two ways. First, the full bipartite metrics are computed over both customer nodes and Site–Product nodes. Second, the Site–Product-subset metrics are computed only over Site–Product nodes. This distinction is essential because the Site–Product-only model predicts only Site–Product nodes. Therefore, the fair comparison between the two graph formulations is made on the common Site–Product node set.

1.7. Research questions

The thesis is guided by the following research questions.

RQ1 **Feasibility and benchmark performance:** Are TGGC-style spectral–temporal graph neural networks competitive with traditional and deep temporal baselines for monthly industrial demand forecasting?

RQ2 **Graph construction:** How does a fixed customer–Site–Product bipartite graph compare with a latent Site–Product graph learned from the time series?

RQ3 **Spectral filter choice:** How does the choice of graph spectral filter affect forecasting performance under the two graph constructions?

These questions reflect both the applied and mathematical parts of the thesis. The applied goal is to forecast sales more accurately. The mathematical goal is to understand how the graph representation and the spectral filter influence the behaviour of the model.

1.8. Contribution of the thesis

The main contributions of the thesis are as follows.

First, the sales forecasting task is cast as a graph signal forecasting problem. This makes it possible to include customer–product relations directly in the model and to test whether these relations are useful for forecasting.

Second, a fixed bipartite graph between customer groups and Site–Product nodes is constructed and tested. This graph is interpretable because edges correspond to observed transaction relationships.

Third, the bipartite graph model is compared with a Site–Product-only TGGC-style model. This comparison separates the effect of the customer side from the effect of graph-based forecasting in general.

Fourth, several graph spectral filters are studied within the same spectral–temporal architecture. This is important because the graph filter determines how information is propagated through the graph. The thesis therefore treats spectral filter selection as a mathematical modelling decision rather than only a tuning detail.

Finally, the graph models are compared with simple traditional baselines and deep learning baselines. This is necessary because time series forecasting research has shown that simple models can remain competitive even against more complex neural architectures [2, 7]. The comparison therefore provides a more honest evaluation of the proposed graph-based approach, especially because a complex neural architecture should be judged against simple persistence-based forecasts.

1.9. Structure of the thesis

The thesis is organized as follows.

Chapter 1 introduces the motivation, background, research questions, and structure of the thesis.

Chapter 2 reviews the related literature. It covers traditional time series forecasting methods, deep learning models for time series, and graph neural networks for multivariate forecasting.

Chapter 3 presents the mathematical background. It explains graph signals, graph Laplacians, spectral graph convolution, temporal filtering, and the graph spectral filters used in the experiments.

Chapter 4 describes the dataset, preprocessing steps, graph construction, node definitions, temporal split, and data characteristics.

Chapter 5 explains the experimental design. It describes the two model formulations, the architecture, the baseline models, the training procedure, and the evaluation protocol.

Chapter 6 presents and discusses the empirical results. It compares the fixed bipartite model, the Site–Product-only model, the spectral filters, the baseline models, and the horizon-wise behaviour.

Chapter 7 summarizes the findings, discusses limitations, and suggests directions for future work.

2. Related Work

2.1. Overview

Time series forecasting has a long research history, starting from classical statistical models and gradually expanding to encompass machine learning, deep learning, attention-based architectures, and graph neural networks. Recent surveys describe the field as increasingly diverse, with no single architecture dominating all forecasting settings [2, 3, 4]. This is important for the present thesis because industrial sales data have several characteristics that make the choice of model nontrivial: temporal dependence, non-stationarity, sparse demand, and relationships between different customer and product series.

The related work in this chapter is organized around the modelling families that are relevant for the experiments. First, classical statistical and traditional machine learning methods are reviewed. Second, deep temporal baselines such as recurrent and convolutional models are discussed. Third, graph neural networks for multivariate and spatio-temporal forecasting are introduced. Finally, spectral-temporal GNNs and graph spectral filters are discussed, since they are the closest methodological foundation for the TGGC-style model used in this thesis.

2.2. Traditional time series forecasting methods

Traditional time series methods remain important because they are simple, interpretable, and often very competitive in practical forecasting tasks. They also provide necessary baselines for evaluating more complex neural models. The survey by Kim et al. notes that statistical models were the dominant tools for time series forecasting before the rise of machine learning and deep learning [2].

2.2.1. Moving average and exponential smoothing

One of the simplest forecasting principles is to use recent observations as the basis for future predictions. A moving average forecast with window length m is

$$\widehat{y}_{t+h} = \frac{1}{m} \sum_{j=0}^{m-1} y_{t-j}.$$

Although very simple, this method is often strong when the series has local persistence and no rapidly changing trend. In the experiments of this thesis, moving-average baselines are included because monthly sales data often have short-memory behaviour.

This baseline is particularly important because it tests whether the forecasting task is mainly driven by recent demand persistence. If a moving average performs strongly, then more complex neural architectures must show that they learn additional structure beyond local temporal smoothing. In sparse or intermittent monthly sales series, however, a moving average may also be difficult to beat because many observations are close to zero or change only slowly from month to month.

Exponential smoothing extends this idea by assigning exponentially decreasing weights to older observations. Early forms of exponential smoothing were developed by Brown, Holt, and Winters [8, 9, 10]. The general idea is that recent observations should receive more weight than older observations. This makes exponential smoothing useful for series where the most recent level is more informative than distant history.

2.2.2. ARIMA and seasonal ARIMA

The Autoregressive Integrated Moving Average model, usually abbreviated as ARIMA, is one of the most widely known statistical time series models. It combines autoregressive terms, differencing, and moving-average terms [5]. An ARIMA model is usually written as

$$\mathrm{ARIMA}(p, d, q),$$

where p is the autoregressive order, d is the differencing order, and q is the moving-average order. The autoregressive part models dependence on past values, the integrated part handles non-stationarity through differencing, and the moving-average part models dependence on past errors.

A simple autoregressive model of order p has the form

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-i} + \varepsilon_t.$$

ARIMA models are useful because they provide a mathematically interpretable framework for modelling autocorrelation. Seasonal ARIMA extends the model by adding seasonal autoregressive, differencing, and moving-average terms. The survey by Kim et al. emphasizes that ARIMA and SARIMA remain important historical foundations for time series forecasting, especially for modelling autocorrelation and seasonality [2].

In this thesis, ARIMA is used as a traditional statistical baseline because it represents a well-established approach for modelling autocorrelation in individual time series. Its relevance lies in testing whether the proposed neural graph models improve over a classical temporal model. However, ARIMA is limited for the present dataset because it is applied independently to each series and does not directly exploit customer–product relations. It may also be difficult to estimate reliably when the available monthly history is short or when demand is sparse and intermittent.

2.2.3. Vector autoregression

For multivariate time series, a natural extension of autoregression is the Vector Autoregression model. If

$$X_t \in \mathbb{R}^N$$

is a vector of N time series at time t, then a VAR model of order p can be written as

$$X_t = c + A_1 X_{t-1} + \cdots + A_p X_{t-p} + \varepsilon_t.$$

VAR models capture linear interactions between variables and are widely used in multivariate time series analysis [11]. They are relevant for this thesis because the sales problem is multivariate: multiple customer and Site–Product series are observed simultaneously. However, VAR models become difficult to estimate when the number of variables is large relative to the number of time points. For this reason, a ridge-regularized VAR baseline is used in the experiments.

This makes the VAR perspective useful conceptually: it motivates the need for models that can represent dependencies between many series without estimating a fully unrestricted interaction matrix from a small number of monthly observations.

2.2.4. Regularized linear models

Ridge regression is a linear model with an L^2 penalty:

$$\min_{\beta} \left\{ \|Y - X\beta\|_2^2 + \lambda \|\beta\|_2^2 \right\}.$$

The penalty stabilizes the regression coefficients when predictors are correlated or when the sample size is limited [12]. In time series forecasting, lagged values can be used as predictors, producing a regularized autoregressive model.

Ridge regression is included as a baseline because it is simple, fast, and often strong when the main structure is approximately linear. It also provides a useful comparison to neural models, because it tests how much is gained by nonlinear modelling and graph structure.

2.3. Deep learning models for time series forecasting

Deep learning methods became popular in time series forecasting because they can learn nonlinear relationships and complex temporal patterns. Surveys of deep learning for time series forecasting describe several important architectural families, including MLPs, recurrent neural networks, convolutional networks, Transformers, and graph neural networks [3, 4, 2].

2.3.1. Recurrent neural networks, LSTM, and GRU

Recurrent neural networks are designed for sequential data. A basic RNN updates a hidden state according to

$$h_t = \sigma(W_x x_t + W_h h_{t-1} + b),$$

where h_t carries information from previous time steps. This structure is suitable for time series because it processes observations in temporal order.

However, standard RNNs suffer from vanishing and exploding gradients when learning long dependencies. Long Short-Term Memory networks were introduced to address this problem through gates and a memory cell [13]. The LSTM architecture uses input, forget, and output gates to control which information is stored, updated, or released. This makes it more effective than a simple RNN for learning longer temporal dependencies.

The Gated Recurrent Unit is a simpler gated recurrent architecture [14]. It combines some of the LSTM gating ideas into a more compact structure with update and reset gates. GRUs often perform similarly to LSTMs while using fewer parameters.

The survey by Kim et al. identifies LSTM and GRU as important developments in recurrent time series modelling, especially because they addressed the limitations of vanilla RNNs [2]. In this thesis, LSTM and GRU are used as deep temporal baselines. They are relevant because they test whether nonlinear sequential modelling alone is sufficient for the forecasting task, without using explicit customer–product graph information. Their limitation in the present setting is that they rely only on temporal patterns in the input series and do not directly encode relations between customers and Site–Product nodes. In addition, recurrent neural networks may be difficult to train effectively when the number of monthly observations is limited.

2.3.2. Convolutional temporal models

Convolutional neural networks were originally developed for spatial pattern extraction, but one-dimensional convolutions can also be applied to time series. A temporal convolution applies filters along the time axis:

$$h_t = \sum_{j=0}^{k-1} w_j x_{t-j}.$$

This allows the model to learn local temporal patterns such as short-term changes or repeated motifs.

Temporal Convolutional Networks use causal and often dilated convolutions to model sequences efficiently [15]. Dilated convolutions increase the receptive field without requiring a very deep model. This makes TCNs useful for sequence modelling and time series forecasting. The survey by Kim et al. describes CNN-based models as useful for extracting local patterns in time series data [2].

In this thesis, a Temporal CNN baseline is used to represent this family of models. It provides a non-recurrent deep learning baseline that captures local temporal patterns in

the 12-month input window. This is relevant because monthly sales may contain short-term changes, local persistence, or repeated motifs that can be captured by convolutional filters. However, like LSTM and GRU, the Temporal CNN does not explicitly model customer–product relationships and therefore cannot directly test the value of graph structure.

2.3.3. Transformer-based forecasting models

Transformers were introduced for sequence modelling using self-attention [16]. In time series forecasting, self-attention allows the model to relate different time steps in the input window. This made Transformers attractive for long sequence forecasting, and many time series Transformer variants were developed, including LogTrans, Informer, Autoformer, Pyraformer, and FEDformer [17, 18, 19, 20, 21].

However, Transformer-based models also have limitations in time series forecasting. Self-attention has quadratic complexity in the sequence length, and recent studies have shown that simple linear models can outperform Transformer models in some long-term forecasting benchmarks [7]. The survey by Kim et al. highlights this as part of a broader diversification of time series forecasting architectures, where researchers increasingly reconsider simple linear models, CNNs, RNNs, GNNs, and hybrid models [2].

Transformers are not used as baselines in the final experiments because the thesis focuses on monthly sales data with a relatively short input window and on the effect of graph structure and spectral filters. Nevertheless, Transformer research is relevant because it shows that more complex architectures are not automatically superior in time series forecasting.

This discussion also motivates a critical point for the empirical design of this thesis: architectural complexity alone is not sufficient to guarantee better forecasting performance. For short monthly sequences, the advantages of attention-based models may be limited, while their parameterization can increase the risk of overfitting. Therefore, Transformers are discussed as an important forecasting family but are not central to the final experiments, which focus instead on the specific question of whether graph structure and spectral filtering improve industrial sales forecasts.

2.4. Graph neural networks for time series

Graph neural networks are designed for data with relational structure. A graph is written as

$$G = (V, E),$$

where V is the node set and E is the edge set. In graph-based time series forecasting, each node has a time-dependent signal. The model must learn both temporal dependencies and relationships between nodes.

The motivation for using GNNs in this thesis is that the forecasting problem is not only temporal but also relational. Customer demand for a Site–Product combination may be connected to other customers, products, or shared purchasing patterns. A graph model

provides a way to represent such dependencies explicitly. This distinguishes graph-based forecasting from LSTM, GRU, CNN, and ARIMA baselines, which mainly model temporal behaviour within individual or aggregated series.

The early graph neural network framework was introduced by Scarselli et al. [22]. Graph Convolutional Networks later became a widely used architecture for learning from graph-structured data [23]. A basic graph convolution aggregates information from neighbouring nodes. In matrix form, a common first-order propagation step is

$$H^{(l+1)} = \sigma(\widetilde{A}H^{(l)}W^{(l)}),$$

where $\widetilde{A}$ is a normalized adjacency matrix, $H^{(l)}$ is the representation at layer l, and $W^{(l)}$ is a learnable weight matrix.

The survey by Kim et al. explains that GNNs became relevant for time series when multivariate data began to be interpreted as variables connected by structural relationships. It also notes that GNNs are particularly useful when interactions between variables matter [2]. This is exactly the case in the present thesis, where customer and Site–Product series are not independent.

2.4.1. Spatio-temporal graph neural networks

Spatio-temporal graph neural networks combine graph modelling with temporal modelling. One important example is the Spatio-Temporal Graph Convolutional Network, which was developed for traffic forecasting [24]. The general idea is to apply graph convolution over the spatial graph and temporal convolution or recurrence over time. This type of architecture is suitable when the data have both network structure and temporal dynamics.

Several later models extended this idea by learning graph structures or using attention mechanisms. Graph Attention Networks use attention weights to determine the importance of neighbouring nodes [25]. Dynamic graph neural networks and temporal graph networks allow the graph or node states to vary over time [26, 27]. The survey by Kim et al. describes these developments as part of the growth of GNN applications in time series forecasting [2].

The models in this thesis belong to the same broad family of spatio-temporal or spectral-temporal GNNs. The difference is that the thesis focuses specifically on graph spectral filters and compares several filter bases within the same forecasting architecture.

At the same time, graph neural networks introduce their own challenges. Their performance depends strongly on the quality of the graph construction, the amount of training data, and the stability of the chosen graph convolution. If the graph is only weakly predictive, or if the available time series are too short and sparse, a GNN may add complexity without improving forecasting accuracy. This is one of the central empirical questions of the thesis.

2.5. Spectral graph convolution

Graph convolution can be defined from a spectral perspective using the graph Laplacian. Let

$$L = I - D^{-1/2}AD^{-1/2}$$

be the normalized graph Laplacian. If

$$L = U\Lambda U^\top$$

is its eigendecomposition, then a spectral graph filter can be written as

$$g_\theta(L)x = Ug_\theta(\Lambda)U^\top x.$$

This is analogous to classical Fourier filtering, but the Fourier basis is replaced by the eigenvectors of the graph Laplacian.

Direct spectral filtering requires eigendecomposition and is expensive for large graphs. Polynomial approximations avoid this by expressing the filter as

$$g_\theta(L)x \approx \sum_{k=0}^{K-1} \theta_k P_k(L)x.$$

Chebyshev polynomial graph convolution was introduced as an efficient localized spectral filtering method [28]. The GCN model of Kipf and Welling can be understood as a first-order approximation of spectral graph convolution [23].

This spectral viewpoint is central to the thesis. The TGGC-style architecture applies a graph spectral filter before the temporal forecasting component. The experiments compare several filter families, including monomial, Gegenbauer, Jacobi, Bernstein, Legendre, Hermite, Laguerre, GCN-type, ARMA, and Cayley filters.

For the present thesis, the spectral perspective is important because it makes the graph filter an explicit modelling choice. Different filters imply different assumptions about how information should propagate across the customer–product graph. Therefore, comparing spectral filters is not only a technical variation, but directly relates to the question of whether the graph structure contains useful forecasting information.

2.6. Rational and polynomial graph filters

Polynomial graph filters are popular because they are computationally efficient and localized. A polynomial filter of order K has the form

$$g_\theta(S) = \sum_{k=0}^{K-1} \theta_k P_k(S),$$

where S is a graph shift operator and P_k is a polynomial basis. Classical orthogonal polynomial families, such as Gegenbauer, Jacobi, Legendre, Hermite, and Laguerre

polynomials, provide structured bases for approximation [29]. The original TGGC model uses a Gegenbauer graph convolution, motivating the inclusion of the Gegenbauer filter in this thesis.

Bernstein polynomial filters have also been studied in graph neural networks. BernNet uses Bernstein approximation to learn flexible graph spectral filters [30]. Bernstein bases are stable on the interval $[0, 1]$, which is why the Bernstein filter in this thesis is applied to a scaled Laplacian.

Rational graph filters are another important class. Unlike finite polynomial filters, rational filters contain inverse terms and can represent more global spectral responses. ARMA graph filters use autoregressive moving-average ideas in the graph spectral domain and can be written in terms of inverse graph operators [31]. CayleyNets use complex rational spectral filters based on the Cayley transform [32]. These filters are relevant because they can express frequency-selective responses more flexibly than low-order polynomial filters.

The comparison between polynomial and rational filters is one of the main methodological elements of this thesis. The experiments show that the best filter depends on the graph construction, which supports the idea that graph filter design should be treated as a mathematical modelling choice.

2.7. TGGC and spectral–temporal GNNs

The model used in this thesis is based on the Temporal Graph Gegenbauer Convolution architecture. TGGC belongs to the class of spectral–temporal GNNs, which combine graph-domain filtering with temporal-domain modelling for time series forecasting [33]. The original TGGC model uses a graph Gegenbauer convolution to increase the expressiveness of graph filtering and combines it with a temporal spectral component.

In the present thesis, the architecture is used as a base framework rather than as a fixed model. The graph operator is changed in two ways. First, a fixed customer–Site–Product bipartite graph is constructed from historical transactions. Second, a latent Site–Product graph is learned from the Site–Product time series. The graph filter is also varied. This leads to a broader formulation: a TGGC-style spectral–temporal GNN with multiple possible graph filters.

This distinction is important. If the thesis only applied TGGC directly, then the main contribution would be empirical. Instead, the thesis studies how the graph representation and spectral filter affect performance. This places the work closer to graph signal processing and spectral graph learning than to a simple application of an existing forecasting model.

2.8. Position of this thesis in the literature

The literature suggests that time series forecasting should not be approached as a simple progression from classical models to increasingly complex neural architectures. Simple models can remain highly competitive, especially when data are short, noisy, or persistent. Deep temporal models provide nonlinear baselines, but they do not automatically solve

the problem of relational dependency between series. Graph neural networks address this relational dimension, but their success depends on whether the graph structure is actually predictive [2]. This thesis is therefore positioned at the intersection of industrial demand forecasting, multivariate time series modelling, and spectral graph neural networks.

The contribution is not the invention of a completely new neural architecture. Rather, the thesis adapts a spectral–temporal GNN to a specific industrial forecasting problem and studies the consequences of different graph constructions. The fixed bipartite graph introduces an interpretable customer–product structure, while the latent Site–Product graph represents learned similarity between aggregated sales series. By comparing these two settings and several spectral filters, the thesis investigates how graph structure and spectral filtering interact in sales forecasting.

This is relevant because multivariate forecasting models often struggle with channel dependency, distribution shift, and domain-specific structure [2]. A graph-based representation provides one way to encode such structure. The results of the thesis therefore contribute to understanding when graph spectral models are useful for industrial sales forecasting and how they compare with simpler temporal baselines.

3. Methods

This chapter presents the mathematical tools used in the thesis. Since the basic forecasting setup and the distinction between univariate and multivariate time series were introduced in Chapter 1, the focus here is on the graph-based part of the model. The chapter is organized around three ideas: graph neural networks, spectral graph theory, and Fourier analysis. These concepts provide the mathematical basis for the TGGC-style spectral–temporal GNN used later in the experiments.

Throughout the chapter, a graph signal is understood as a vector of values defined on the nodes of a graph. In the fixed bipartite formulation, the nodes are customer groups and Site–Product nodes. In the Site–Product-only formulation, the nodes are only Site–Product series. This distinction is important: the mathematics of graph filtering is the same in both cases, but the meaning of the graph operator changes with the graph construction.

3.1. Notation

Table 3.1 summarizes the notation used in the mathematical and experimental chapters.

Table 3.1.: Main notation used in the thesis.

Symbol	Meaning
$G = (V, E, A)$	Weighted graph with node set V, edge set E, and adjacency matrix A
V_c	Set of customer nodes
V_p	Set of Site–Product nodes
N	Number of graph nodes
A	Weighted adjacency matrix
B	Bipartite customer–Site–Product interaction block
D	Degree matrix
L	Normalized graph Laplacian
$S = I - L$	Graph shift operator
$x \in \mathbb{R}^N$	Graph signal at a fixed time point
$X_t \in \mathbb{R}^N$	Multivariate graph signal at month t
U	Matrix of graph Laplacian eigenvectors
Λ	Diagonal matrix of graph Laplacian eigenvalues
W	Lookback window length
H	Forecast horizon
K	Graph filter order

3.2. Graph Neural Networks

Graph neural networks are neural models for data defined on graphs. In this thesis, the graph represents structural relations between sales entities. In the fixed bipartite formulation, the two node types are customer groups and Site–Product nodes. In the Site–Product-only formulation, the nodes are only Site–Product series. The purpose of this section is to introduce the mathematical form of graph neural networks and to connect message passing with the spectral graph filters used later in the thesis.

Definition 3.1 (Weighted graph)**.** A weighted graph is a triple

$$G = (V, E, A),$$

where

$$V = \{v_1, \dots, v_N\}$$

is the node set, $E \subseteq V \times V$ is the edge set, and

$$A \in \mathbb{R}^{N \times N}$$

is the weighted adjacency matrix. The entry A_{ij} represents the weight of the edge from node v_j to node v_i. If the graph is undirected, then $A = A^\top$.

Definition 3.2 (Graph signal)**.** Let $G = (V, E, A)$ be a graph with N nodes. A graph signal is a function

$$x : V \to \mathbb{R}.$$

After fixing an ordering of the nodes, the graph signal can be written as a vector

$$x = (x_1, \dots, x_N)^\top \in \mathbb{R}^N.$$

More generally, a graph signal with d features per node is represented by

$$X \in \mathbb{R}^{N \times d}.$$

In the forecasting problem of this thesis, the graph signal is time-dependent. At month t, the graph signal is denoted by

$$X_t \in \mathbb{R}^N.$$

The sequence

$$(X_1, \dots, X_T)$$

is therefore a multivariate time series indexed by the nodes of a graph.

3.2.1. Message passing

The most common formulation of graph neural networks is based on message passing. A node representation is updated by aggregating information from neighbouring nodes. This formalism covers many modern GNN architectures [34, 35].

Definition 3.3 (Neighbourhood)**.** For a node $v_i \in V$, its neighbourhood is

$$\mathcal{N}(i) = \{j \in \{1, \dots, N\} : (v_j, v_i) \in E\}.$$

If self-loops are included, the extended neighbourhood is

$$\overline{\mathcal{N}}(i) = \mathcal{N}(i) \cup \{i\}.$$

Definition 3.4 (Message-passing layer)**.** Let

$$h_i^{(\ell)} \in \mathbb{R}^{d_\ell}$$

be the representation of node i at layer ℓ. A message-passing layer is defined by

$$m_i^{(\ell+1)} = \mathrm{AGG}^{(\ell+1)}\left(\left\{\mathrm{MSG}^{(\ell+1)}\left(h_i^{(\ell)}, h_j^{(\ell)}, e_{ji}\right) : j \in \mathcal{N}(i)\right\}\right),$$

and

$$h_i^{(\ell+1)} = \mathrm{UPD}^{(\ell+1)}\left(h_i^{(\ell)}, m_i^{(\ell+1)}\right).$$

Here e_{ji} denotes possible edge information, MSG is a message function, AGG is a neighbourhood aggregation function, and UPD is a node update function.

Definition 3.5 (Permutation-invariant aggregation)**.** An aggregation function

$$\mathrm{AGG} : \{z_1, \dots, z_m\} \mapsto z$$

is permutation invariant if for every permutation π of $\{1, \dots, m\}$,

$$\mathrm{AGG}(\{z_1, \dots, z_m\}) = \mathrm{AGG}(\{z_{\pi(1)}, \dots, z_{\pi(m)}\}).$$

Proposition 3.6 (Permutation invariance of standard aggregators)**.** *The functions*

$$\mathrm{sum}(\{z_j\}) = \sum_j z_j,$$

$$\mathrm{mean}(\{z_j\}) = \frac{1}{m}\sum_j z_j,$$

and

$$\max(\{z_j\}) = \max_j z_j$$

are permutation invariant.

Proof. Let π be a permutation of $\{1, \dots, m\}$. Since addition is commutative,

$$\sum_{j=1}^{m} z_{\pi(j)} = \sum_{j=1}^{m} z_j.$$

Therefore, the sum aggregator is permutation invariant. The mean aggregator is the sum aggregator multiplied by the constant $1/m$, so it is also permutation invariant. The maximum over a finite set is independent of the ordering of the elements, so the max aggregator is permutation invariant. □

Permutation invariance is necessary because the neighbours of a node do not have a canonical order. In the sales graph, the set of customers connected to a Site–Product node is unordered. A model should not change its prediction if the same customers are listed in a different order.

3.2.2. Matrix form of graph propagation

A simple weighted message-passing layer can be written as

$$h_i^{(\ell+1)} = \sigma\left(W_0^{(\ell)} h_i^{(\ell)} + \sum_{j \in \mathcal{N}(i)} A_{ij} W_1^{(\ell)} h_j^{(\ell)}\right).$$

In matrix form, this becomes

$$H^{(\ell+1)} = \sigma\left(AH^{(\ell)} W^{(\ell)}\right),$$

where

$$H^{(\ell)} = \begin{pmatrix} (h_1^{(\ell)})^\top \\ \vdots \\ (h_N^{(\ell)})^\top \end{pmatrix} \in \mathbb{R}^{N \times d_\ell}.$$

In practice, A is often replaced by a normalized adjacency matrix. Let

$$D = \operatorname{diag}(d_1, \dots, d_N), \qquad d_i = \sum_{j=1}^{N} A_{ij}.$$

The symmetrically normalized adjacency is

$$S = D^{-1/2} A D^{-1/2}.$$

Then the propagation rule is

$$H^{(\ell+1)} = \sigma\left(SH^{(\ell)} W^{(\ell)}\right).$$

Proposition 3.7 (Degree-normalized propagation)**.** *Let $S = D^{-1/2}AD^{-1/2}$. Then the contribution of node j to node i in SH is weighted by*

$$S_{ij} = \frac{A_{ij}}{\sqrt{d_i d_j}}.$$

Proof. By definition,

$$S_{ij} = (D^{-1/2}AD^{-1/2})_{ij}.$$

Since $D^{-1/2}$ is diagonal,

$$S_{ij} = D_{ii}^{-1/2} A_{ij} D_{jj}^{-1/2} = \frac{A_{ij}}{\sqrt{d_i d_j}}.$$

□

Remark 3.8. Degree normalization is important in the fixed bipartite graph because Site–Product nodes may be connected to many customers, while many customer nodes are connected to only a few Site–Product nodes. Without normalization, high-degree nodes could dominate the propagation.

3.2.3. Graph convolutional networks

The Graph Convolutional Network is one of the standard feedforward GNN architectures [23]. It adds self-loops and applies a normalized first-order propagation.

Let

$$\widehat{A} = A + I$$

and let

$$\widehat{D}_{ii} = \sum_j \widehat{A}_{ij}.$$

A GCN layer is

$$H^{(\ell+1)} = \sigma\left(\widehat{D}^{-1/2}\widehat{A}\widehat{D}^{-1/2}H^{(\ell)}W^{(\ell)}\right).$$

Remark 3.9. The GCN layer can be interpreted both as a message-passing layer and as a first-order approximation of spectral graph convolution. This connection is important because the model studied in this thesis generalizes this idea by replacing the first-order filter with richer polynomial and rational graph filters.

3.2.4. Other message-passing architectures

Several GNN architectures differ mainly in the choice of aggregation and weighting mechanism.

Definition 3.10 (GraphSAGE mean aggregation)**.** The mean-aggregation version of GraphSAGE is

$$h_i^{(\ell+1)} = \sigma\left(W^{(\ell)}\left[h_i^{(\ell)} \,\|\, \frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)} h_j^{(\ell)}\right]\right),$$

where $\|$ denotes concatenation [36].

Definition 3.11 (Graph attention layer)**.** A graph attention layer has the form

$$h_i^{(\ell+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(\ell)} W^{(\ell)} h_j^{(\ell)}\right),$$

where

$$\alpha_{ij}^{(\ell)} = \frac{\exp(a^{(\ell)}(Wh_i^{(\ell)}, Wh_j^{(\ell)}))}{\sum_{r\in\mathcal{N}(i)} \exp(a^{(\ell)}(Wh_i^{(\ell)}, Wh_r^{(\ell)}))}.$$

The coefficients $\alpha_{ij}^{(\ell)}$ are learned attention weights [25].

Definition 3.12 (Graph Isomorphism Network layer)**.** A Graph Isomorphism Network layer is

$$h_i^{(\ell+1)} = \mathrm{MLP}^{(\ell)}\left((1+\epsilon^{(\ell)})h_i^{(\ell)} + \sum_{j\in\mathcal{N}(i)} h_j^{(\ell)}\right).$$

The summation aggregator is used because of its strong ability to distinguish multisets [37].

These architectures are not the main experimental models of this thesis, but they show the generality of the message-passing framework. The implemented model belongs to the broader GNN family, but it uses spectral graph filters rather than a standard neighbourhood aggregation layer.

3.2.5. Bipartite graph propagation

The main graph proposed in this thesis is bipartite. The node set is

$$V = V_c \cup V_p, \qquad V_c \cap V_p = \varnothing,$$

where V_c denotes customer nodes and V_p denotes Site–Product nodes. The adjacency matrix is

$$A = \begin{pmatrix} 0 & B \\ B^\top & 0 \end{pmatrix}.$$

Let the node representation be written as

$$h = \begin{pmatrix} h_c \\ h_p \end{pmatrix},$$

where h_c is the customer-side signal and h_p is the Site–Product-side signal.

Proposition 3.13 (One-step bipartite propagation)**.** *For the bipartite adjacency matrix*

$$A = \begin{pmatrix} 0 & B \\ B^\top & 0 \end{pmatrix},$$

one propagation step gives

$$Ah = \begin{pmatrix} Bh_p \\ B^\top h_c \end{pmatrix}.$$

Thus, one graph step transfers information from one node type to the other.

Proof. By block matrix multiplication,

$$Ah = \begin{pmatrix} 0 & B \\ B^\top & 0 \end{pmatrix} \begin{pmatrix} h_c \\ h_p \end{pmatrix} = \begin{pmatrix} 0h_c + Bh_p \\ B^\top h_c + 0h_p \end{pmatrix} = \begin{pmatrix} Bh_p \\ B^\top h_c \end{pmatrix}.$$

□

Proposition 3.14 (Two-step bipartite propagation)**.** *For the same bipartite adjacency matrix,*

$$A^2 h = \begin{pmatrix} BB^\top h_c \\ B^\top B h_p \end{pmatrix}.$$

Thus, two graph steps induce within-type propagation.

Proof. First,

$$A^2 = \begin{pmatrix} 0 & B \\ B^\top & 0 \end{pmatrix} \begin{pmatrix} 0 & B \\ B^\top & 0 \end{pmatrix} = \begin{pmatrix} BB^\top & 0 \\ 0 & B^\top B \end{pmatrix}.$$

Multiplying by h gives

$$A^2 h = \begin{pmatrix} BB^\top h_c \\ B^\top B h_p \end{pmatrix}.$$

□

Remark 3.15. The matrix $BB^\top$ represents induced customer–customer similarity through shared Site–Product neighbours. Similarly, $B^\top B$ represents induced Site–Product similarity through shared customer neighbours. Therefore, even though the fixed graph has no direct customer–customer or Site–Product–Site–Product edges, such relations appear through even powers of the bipartite adjacency.

The same reasoning applies to the normalized adjacency

$$S = D^{-1/2} A D^{-1/2} = \begin{pmatrix} 0 & \widetilde{B} \\ \widetilde{B}^\top & 0 \end{pmatrix}.$$

Then

$$S \begin{pmatrix} h_c \\ h_p \end{pmatrix} = \begin{pmatrix} \widetilde{B} h_p \\ \widetilde{B}^\top h_c \end{pmatrix},$$

and

$$S^2 \begin{pmatrix} h_c \\ h_p \end{pmatrix} = \begin{pmatrix} \widetilde{B}\widetilde{B}^\top h_c \\ \widetilde{B}^\top \widetilde{B} h_p \end{pmatrix}.$$

This is the main mathematical reason for using a bipartite graph in the thesis. Customer information can influence Site–Product forecasts through structured graph propagation, while Site–Product relations can be induced through shared customer behaviour.

3.2.6. Relation between message passing and spectral filtering

Message passing and spectral graph filtering are closely related. Let S be a graph shift operator, for example

$$S = D^{-1/2} A D^{-1/2}.$$

A polynomial graph filter of order K has the form

$$g_\theta(S) = \sum_{k=0}^{K-1} \theta_k S^k.$$

Applied to a graph signal x, this gives

$$g_\theta(S)x = \sum_{k=0}^{K-1} \theta_k S^k x.$$

Proposition 3.16 (Polynomial filters as multi-hop propagation)**.** *Let S be a graph shift operator. The term $S^k x$ depends only on nodes connected by walks of length at most k. Therefore, a polynomial graph filter of order K aggregates information from at most $(K-1)$-hop neighbourhoods.*

Proof. For $k = 1$, the vector Sx has entries

$$(Sx)_i = \sum_j S_{ij} x_j.$$

Thus, $(Sx)_i$ depends only on nodes j with $S_{ij} \neq 0$, i.e. one-hop neighbours. Assume $S^k x$ depends only on nodes reachable within k steps. Then

$$S^{k+1}x = S(S^k x).$$

The value at node i is a weighted combination of values of $S^k x$ at one-hop neighbours of i. Each of those values depends on nodes at distance at most k from that neighbour. Therefore, $S^{k+1}x$ depends on nodes at distance at most $k+1$ from i. The result follows by induction. □

Remark 3.17. This proposition connects spectral filtering to message passing. Polynomial spectral filters can be interpreted as weighted combinations of multi-hop message-passing operations. The spectral viewpoint is stronger because it also describes how different graph frequencies are amplified or suppressed.

3.2.7. GNNs for multivariate time series

In multivariate time series forecasting, the graph signal changes over time:

$$X_t \in \mathbb{R}^N.$$

A graph-based forecasting model has the general form

$$\widehat{X}_{t+1:t+H} = f_\Theta(X_{t-W+1:t}, G).$$

The graph G determines how information is shared across variables, while the temporal component determines how past values are transformed into future predictions.

In spatio-temporal graph neural networks, this is often implemented by combining graph convolution and temporal convolution or recurrence [24]. The model used in this thesis follows the same general principle but uses spectral graph filtering and temporal Fourier filtering:

$$X \longmapsto \text{graph spectral filtering} \longmapsto \text{temporal Fourier filtering} \longmapsto \widehat{Y}.$$

3.2.8. Limitations of standard message passing

Standard message passing has several known limitations.

Definition 3.18 (Oversmoothing)**.** Oversmoothing refers to the phenomenon where node representations become increasingly similar as the number of graph propagation layers increases.

In many graph convolutional models, repeated propagation behaves like repeated smoothing over the graph. As depth increases, node embeddings may converge toward a low-dimensional subspace and lose node-specific information [38, 39].

Definition 3.19 (Oversquashing)**.** Oversquashing refers to the compression of information from a large neighbourhood into a fixed-dimensional node representation.

If many distant nodes influence a target node through a small number of graph paths, their information may be compressed too strongly. This is a known limitation of message-passing GNNs [40].

Remark 3.20. These limitations are relevant for the fixed bipartite sales graph. Many customers may influence a Site–Product node, but the temporal sample size is limited and the graph is sparse. Therefore, the thesis uses graph spectral filters rather than simply increasing the number of message-passing layers.

3.2.9. Role of GNNs in this thesis

The role of GNNs in this thesis is to provide a framework for relational sales forecasting. The graph defines which node signals are related, and graph propagation defines how information is exchanged.

In the fixed bipartite model,

$$V = V_c \cup V_p,$$

and information propagates through customer–Site–Product relations. In the Site–Product-only model,

$$V = V_p,$$

and the graph is learned from the Site–Product time series. Both settings use the same general idea:

node-level forecasting with graph-based information sharing.

The following chapters specialize this general GNN framework to spectral graph theory and Fourier analysis. The graph operator determines the graph frequencies, the spectral filter determines the graph-domain response, and the temporal Fourier block models the time-domain structure.

3.3. Spectral Graph Theory

Spectral graph theory studies graphs through matrices associated with them, especially the adjacency matrix and the graph Laplacian. It provides the mathematical basis for graph Fourier analysis and spectral graph convolution [41, 42].

3.3.1. Graph Laplacians

Definition 3.21 (Combinatorial graph Laplacian)**.** Let $G = (V, E, A)$ be an undirected weighted graph with degree matrix D. The combinatorial graph Laplacian is

$$L_{\text{comb}} = D - A.$$

Definition 3.22 (Normalized graph Laplacian)**.** Assume that all node degrees are positive. The normalized graph Laplacian is

$$L = I - D^{-1/2} A D^{-1/2}.$$

The normalized Laplacian is used throughout this thesis because it reduces the influence of node degree differences. This is especially important for the fixed bipartite graph, where Site–Product nodes can have many customer neighbours while many customer nodes have only a small number of Site–Product neighbours.

Proposition 3.23 (Basic properties of the normalized Laplacian)**.** *Let $G = (V, E, A)$ be an undirected weighted graph with nonnegative edge weights and positive degrees. Then*

$$L = I - D^{-1/2} A D^{-1/2}$$

is symmetric, positive semidefinite, and has all eigenvalues in the interval $[0, 2]$.

Proof. Since A is symmetric and $D^{-1/2}$ is diagonal, the matrix $D^{-1/2}AD^{-1/2}$ is symmetric. Therefore L is symmetric. For any $x \in \mathbb{R}^N$, set $y = D^{-1/2}x$. Then

$$x^\top Lx = y^\top Dy - y^\top Ay.$$

Using $d_i = \sum_j A_{ij}$ and the symmetry of A, this becomes

$$y^\top Dy - y^\top Ay = \frac{1}{2}\sum_{i,j=1}^{N} A_{ij}(y_i - y_j)^2 \geq 0.$$

Thus L is positive semidefinite. The spectral bound $\lambda \in [0, 2]$ for the normalized Laplacian is a standard result in spectral graph theory [41, 42]. □

Because L is real and symmetric, it admits an orthonormal eigendecomposition:

$$L = U\Lambda U^\top,$$

where

$$U = (u_0, \ldots, u_{N-1})$$

is an orthogonal matrix of eigenvectors and

$$\Lambda = \mathrm{diag}(\lambda_0, \ldots, \lambda_{N-1})$$

contains the eigenvalues. The eigenvectors play the role of graph Fourier modes, and the eigenvalues play the role of graph frequencies.

3.3.2. Graph shift operator

In the experiments, many filters are written in terms of the graph shift

$$S = I - L = D^{-1/2}AD^{-1/2}.$$

If the spectrum of L lies in $[0, 2]$, then the spectrum of $S = I - L$ lies in $[-1, 1]$. This interval is natural for several orthogonal polynomial families, including Gegenbauer, Jacobi, and Legendre polynomials.

For the fixed bipartite graph,

$$S = \begin{pmatrix} 0 & \widehat{B} \\ \widehat{B}^\top & 0 \end{pmatrix},$$

where $\widehat{B}$ is the degree-normalized bipartite block. Therefore

$$S \begin{pmatrix} x_c \\ x_p \end{pmatrix} = \begin{pmatrix} \widehat{B}x_p \\ \widehat{B}^\top x_c \end{pmatrix},$$

and

$$S^2 \begin{pmatrix} x_c \\ x_p \end{pmatrix} = \begin{pmatrix} \widehat{B}\widehat{B}^\top x_c \\ \widehat{B}^\top \widehat{B}x_p \end{pmatrix}.$$

This explains why odd powers of S transfer information across the two sides of the bipartite graph, while even powers describe within-type relations induced by shared neighbours.

3.3.3. Spectral graph filters

A spectral graph filter is a function of the graph Laplacian or graph shift. If $L = U\Lambda U^\top$, then a filter g acts on a graph signal x by

$$g(L)x = Ug(\Lambda)U^\top x,$$

where

$$g(\Lambda) = \mathrm{diag}(g(\lambda_0), \ldots, g(\lambda_{N-1})).$$

This is the graph analogue of filtering a classical signal in the Fourier domain [42].

Direct computation of U can be expensive and unstable for large graphs. A common solution is to approximate the filter by a polynomial in the graph operator:

$$g(S)x \approx \sum_{k=0}^{K-1} \theta_k P_k(S)x,$$

where P_k are basis polynomials and θ_k are trainable coefficients. Chebyshev polynomial graph filters are a standard example of this idea [28]. The first-order GCN filter can also be viewed as a simplified spectral approximation [23].

The TGGC-style architecture used in this thesis follows the same general principle but compares several filter bases. The detailed experimental list of filters is given in Chapter 5. Mathematically, they can be grouped into polynomial filters and rational filters.

Polynomial filters have the form

$$g_{\mathrm{poly}}(S) = \sum_{k=0}^{K-1} \theta_k P_k(S).$$

They are local in the graph domain because powers of the graph operator aggregate information over graph neighbourhoods. Rational filters include inverse operators. Two important examples are

$$g_{\mathrm{ARMA}}(S) = \sum_{k=0}^{K-1} \theta_k (I - \alpha S)^{-1} S^k$$

and the Cayley transform

$$C(S) = (I - hS)^{-1}(I + hS).$$

ARMA graph filters and Cayley filters have been proposed as flexible rational spectral filters for graph neural networks [31, 32]. Their relevance in this thesis is empirical as well as mathematical: the best filter differs between the fixed bipartite graph and the latent Site–Product graph.

3.4. Fourier Analysis

Fourier analysis appears twice in the model. First, graph Fourier analysis describes signals in terms of graph frequencies. Second, temporal Fourier analysis describes the evolution of each node signal in terms of temporal frequencies. The spectral–temporal model combines these two views.

3.4.1. Graph Fourier transform

Let

$$L = U\Lambda U^\top$$

be the eigendecomposition of the normalized graph Laplacian. The graph Fourier transform of a signal $x \in \mathbb{R}^N$ is

$$\widehat{x} = U^\top x.$$

The inverse graph Fourier transform is

$$x = U\widehat{x}.$$

The coefficient $\widehat{x}_k$ measures the contribution of the k-th graph Fourier mode u_k to the signal x. Small eigenvalues correspond to smooth graph modes, while larger eigenvalues correspond to modes that vary more strongly across connected nodes [42].

This interpretation is important for spectral graph filtering. A graph filter does not only mix neighbouring nodes; it modifies the signal according to graph frequency. For example, a low-pass filter suppresses high-frequency variation over the graph, while a more expressive filter can amplify or suppress different spectral regions.

3.4.2. Temporal discrete Fourier transform

For a finite temporal signal

$$z = (z(0), \dots, z(T-1)) \in \mathbb{R}^T,$$

the discrete Fourier transform is

$$\widetilde{z}(k) = \sum_{t=0}^{T-1} z(t) e^{-2\pi i k t/T}, \qquad k = 0, \dots, T-1.$$

The inverse transform is

$$z(t) = \frac{1}{T} \sum_{k=0}^{T-1} \widetilde{z}(k) e^{2\pi i k t/T}.$$

The DFT decomposes a finite sequence into oscillatory components. In monthly sales data, temporal frequencies may correspond to slowly varying levels, seasonal patterns, or short-term fluctuations.

In practice, the model uses the real Fourier transform along the temporal dimension. Since the input window is short, all available temporal modes are used in the final implementation rather than selecting a random subset. This keeps the temporal component fixed across graph-filter experiments and makes the filter comparison more stable.

3.4.3. Spectral–temporal filtering

The model architecture combines graph-frequency and time-frequency operations. At a high level, the input signal is a matrix

$$X \in \mathbb{R}^{W \times N},$$

where W is the input length and N is the number of graph nodes. The graph spectral part filters across nodes, while the temporal Fourier part filters across the time dimension.

Conceptually, this can be written as

$$X \longmapsto g(S)X \longmapsto \mathcal{F}_t(g(S)X) \longmapsto \mathcal{F}_t^{-1}(\cdot) \longmapsto \widehat{Y},$$

where $g(S)$ is the graph spectral filter, $\mathcal{F}_t$ is the temporal Fourier transform, and $\widehat{Y} \in \mathbb{R}^{H \times N}$ is the forecast. This is the mathematical idea behind spectral–temporal GNNs such as TGGC [33].

The key point is that the model separates two kinds of structure. Graph spectral filtering handles relations between nodes, such as customer–product connections or learned Site–Product similarity. Temporal Fourier filtering handles oscillatory or seasonal behaviour over time. The experiments in Chapter 5 keep the temporal component fixed and compare how different graph operators and graph spectral filters affect forecasting behaviour.

3.5. Connection to the Experimental Model

The methods introduced in this chapter lead directly to the experimental model. The graph neural network viewpoint explains why related sales series can exchange information. Spectral graph theory provides the graph Laplacian, graph shift, and graph Fourier basis. Fourier analysis provides the frequency-domain interpretation used both over the graph and over time.

The fixed bipartite model uses the normalized Laplacian of the customer–Site–Product graph. The Site–Product-only model uses a graph learned from the Site–Product series. In both cases, the forecasting architecture applies a graph spectral filter followed by temporal Fourier filtering. The next chapters describe how the data are prepared, how the graph operators are constructed, and how the spectral filters are evaluated empirically.

4. Data

4.1. Data source and forecasting target

The empirical study is based on monthly sales data from an industrial packaging business. Each row in the raw data represents a sales observation associated with a customer group, a site, a product category, and a monthly time period. The forecasting target is

`Net Sales K ACT`,

which is measured in thousands of sales units. In the experiments this variable is denoted by $x_{t,i}$, where t is the month and i is the node index.

The sales variable was kept with its original sign. This is important because the data may contain credit notes, returns, or corrections. Negative values therefore carry information about the sales process and should not be removed from the forecasting target. However, the graph edge weights must be nonnegative. For this reason, the node signals preserve signed sales values, while the graph edges are constructed from historical absolute transaction volume.

Before modelling, the dataset was cleaned by removing one very small and unreliable site and by excluding product categories that were marked as non-standard or not meaningful for the final forecasting task. This cleaning step was performed before the graph construction. The final data therefore represent the cleaned sales universe used consistently in the bipartite and Site–Product experiments.

4.2. Temporal aggregation

The data were aggregated at monthly frequency. Let

$$t = 1, \ldots, T$$

denote the sequence of months. For each node i, the monthly signal is defined as the sum of signed sales values assigned to that node during month t:

$$x_{t,i} = \sum_{\ell \in \mathcal{D}_{t,i}} \text{sales}_\ell,$$

where $\mathcal{D}_{t,i}$ is the set of sales records in month t associated with node i. Missing node–month combinations are filled with zero, because no recorded transaction for a node in a month means zero observed sales for that node in the aggregated matrix.

The final monthly range used by the experiment is

$$2023 - 01 \text{ to } 2026 - 02.$$

To prepare the data for supervised learning, the time series is divided into sliding windows. The specific input and forecast horizons are defined in the experimental setup (Section 5.2).

4.3. Node definitions

Two graph representations are used in the thesis. The first is the fixed bipartite graph, which contains customer nodes and Site–Product nodes. The second is the Site–Product-only graph, which contains only Site–Product nodes.

A Site–Product node is defined by combining the site and product category description:

$$\text{Site–Product} = \texttt{SITE} + \texttt{product\_category\_description1}.$$

The fixed bipartite graph contains

$$|V_c| = 965$$

customer nodes and

$$|V_p| = 28$$

Site–Product nodes. Hence, the total number of nodes in the bipartite graph is

$$|V_c| + |V_p| = 993.$$

The Site–Product-only experiment uses only the 28 Site–Product nodes.

Table 4.1.: Node and graph size after preprocessing.

Quantity	Value
Customer nodes	965
Site–Product nodes	28
Total fixed bipartite nodes	993
Customer–Site–Product edges	1,275
Connected customers in the fixed graph	942
Connected Site–Product nodes in the fixed graph	28
Bipartite block density	0.0472

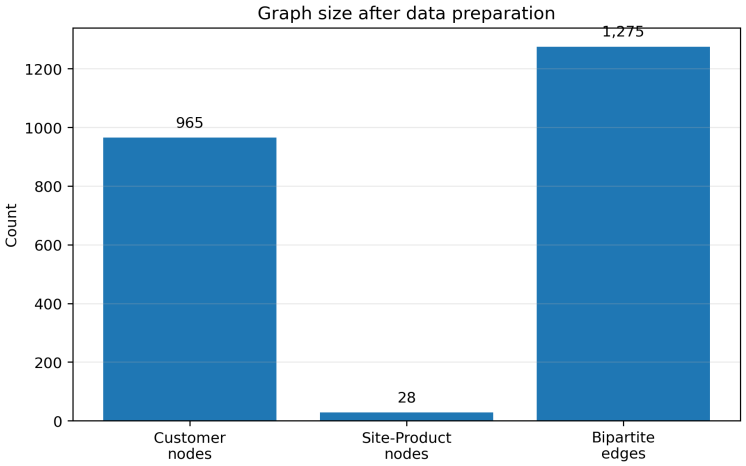

Figure 4.1.: Size of the fixed bipartite graph after preprocessing.

4.4. Fixed bipartite graph construction

The fixed bipartite graph is constructed from historical customer–Site–Product interactions. Let V_c be the set of customer nodes and V_p be the set of Site–Product nodes. The bipartite adjacency matrix has the form

$$A = \begin{pmatrix} 0 & B \\ B^\top & 0 \end{pmatrix},$$

where $B \in \mathbb{R}_+^{|V_c| \times |V_p|}$ is the customer–Site–Product interaction matrix.

For customer c and Site–Product node p, the edge weight is constructed from historical absolute sales volume:

$$B_{cp} = \log\left(1 + \sum_{\ell \in \mathcal{D}_{c,p}^{\text{train}}} |\text{sales}_\ell|\right).$$

The logarithm is used to reduce the effect of very large customers or very large product groups. Only the training-visible history is used to construct the graph, so that the edge weights do not use information from the test period.

The graph contains $1,275$ observed customer–Site–Product edges. Since the full bipartite block has

$$965 \times 28$$

possible customer–Site–Product pairs, the graph is sparse. The density of the bipartite block is approximately

$$0.0472.$$

This sparsity is an important feature of the data and motivates the comparison between the fixed bipartite graph and the smaller Site–Product-only graph.

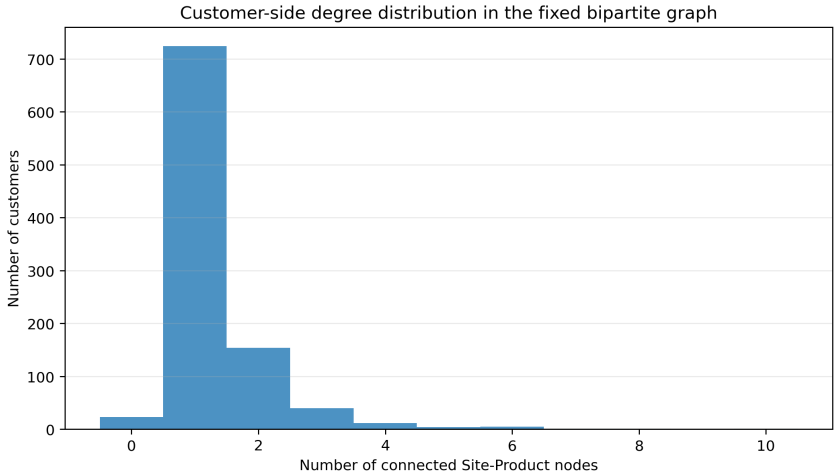

Figure 4.2.: Customer-side degree distribution in the fixed bipartite graph.

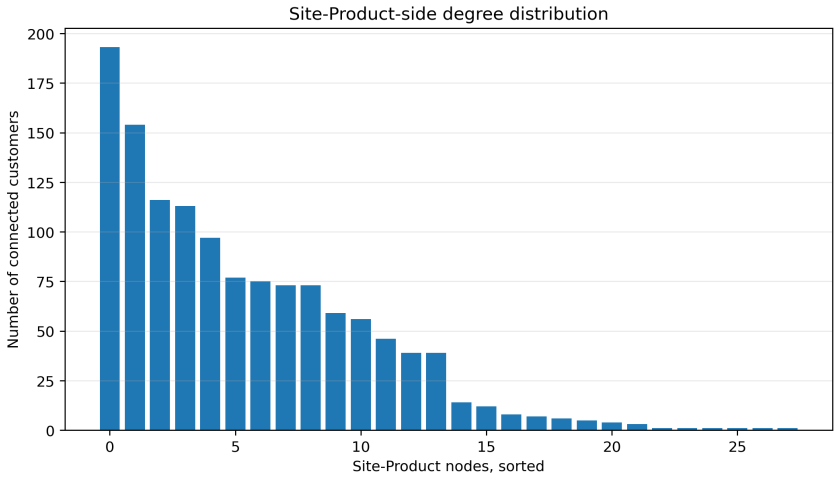

Figure 4.3.: Site–Product-side degree distribution in the fixed bipartite graph.

The degree distributions show the asymmetry of the graph. There are many customer nodes but only a small number of Site–Product nodes. A customer can be connected to only a few Site–Product nodes, while a Site–Product node can be connected to many customers. This asymmetry is one of the reasons why the full bipartite forecasting task is harder than the Site–Product-only task.

4.5. Site–Product matrix

The Site–Product-only matrix is obtained by aggregating sales only at the Site–Product level. If $p \in V_p$, then

$$x_{t,p} = \sum_{\ell \in \mathcal{D}_{t,p}} \text{sales}_\ell.$$

This gives a smaller matrix

$$X^{(p)} \in \mathbb{R}^{T \times |V_p|},$$

with 28 nodes. The Site–Product graph is therefore much smaller and less sparse than the fixed bipartite graph. It is also closer to the final business forecasting target because it removes customer-level sparsity.

The Site–Product-only experiment is not a replacement for the bipartite model. Rather, it is used as a comparison. It tests whether the explicit customer–product structure in the bipartite graph helps, weakens, or approximately matches the forecasting performance obtained from a smaller latent Site–Product graph.

4.6. Trend and sparsity

Figure 4.4 shows the monthly sales trend over the final evaluation period. The customer and Site–Product totals are very close because they are two aggregations of the same underlying transactions: once by customer and once by Site–Product. The figure is therefore mainly used to show the scale and variation of the evaluated months.

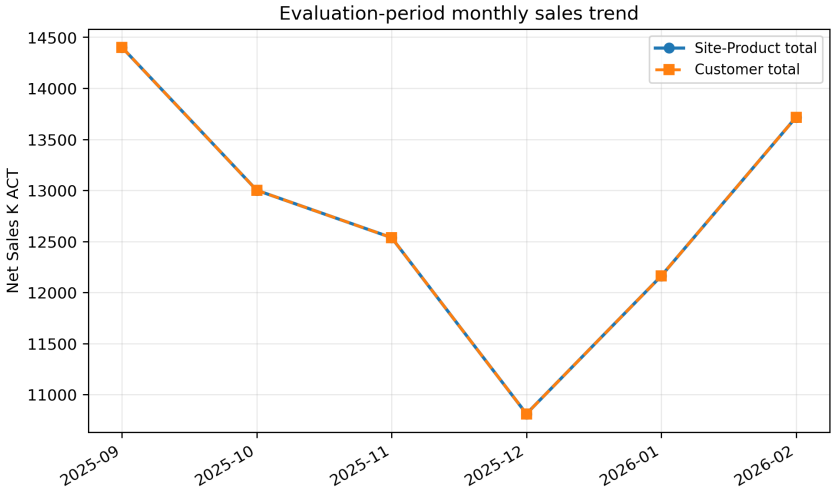

Figure 4.4.: Monthly sales trend over the evaluation period.

The main difficulty in the fixed bipartite graph is not the Site–Product side, but the customer side. During the evaluation period, many customer nodes have no observed

sales at all, while most Site–Product nodes are active in several months. This is shown in Figure 4.5.

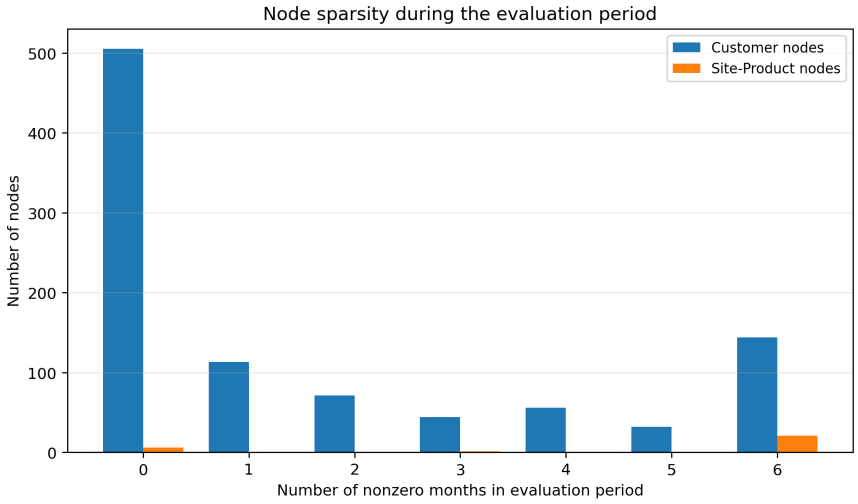

Figure 4.5.: Number of nonzero months per node during the evaluation period.

In the evaluation period, the average number of nonzero months is approximately

$$1.69$$

for customer nodes and

$$4.61$$

for Site–Product nodes. In addition,

$$501$$

customer nodes and

$$6$$

Site–Product nodes have zero total absolute sales during the evaluation period. This explains why percentage-based errors are much more unstable for customer nodes than for Site–Product nodes.

4.7. Training, validation, and test split

The split is chronological. The forecasting windows are divided into training, validation, and test sets using the ratios

$$70\%, \qquad 15\%, \qquad 15\%.$$

This gives

$$16$$

training windows,

$$4$$

validation windows, and

$$4$$

test windows. Because the task is multi-step forecasting, consecutive samples have overlapping target months. The target periods covered by the split are summarized in Table 4.2.

Table 4.2.: Chronological split used in the final experiments.

Split	Number of windows	First target month	Last target month
Training	16	2024-01	2025-06
Validation	4	2025-05	2025-10
Test	4	2025-09	2026-02

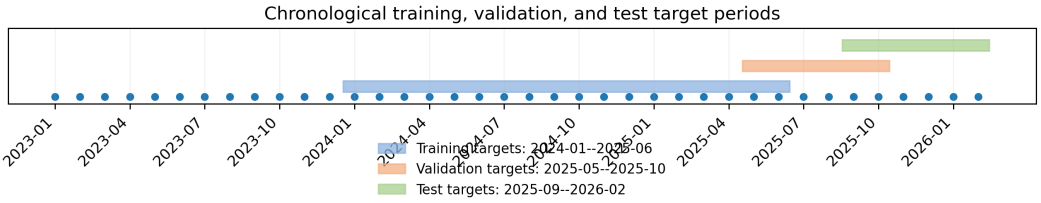

Figure 4.6.: Chronological split of the target periods.

To avoid information leakage, both the normalization statistics and the fixed bipartite edge weights are computed only from the training-visible period. The normalization statistics are computed from

$$2023-01 \text{ to } 2025-06.$$

The validation and test periods are not used for normalization or for the fixed graph construction. After training, predictions are transformed back to the original scale before MAE, RMSE, and WAPE are calculated.

4.8. Summary of the prepared data

The prepared data have three important properties that affect the modelling results.

First, the graph is highly asymmetric. There are many customer nodes and only a small number of Site–Product nodes. This makes the fixed bipartite graph structurally meaningful but also sparse.

Second, the Site–Product series are more aggregated and smoother than the customer series. This makes the Site–Product-only model a useful comparison and explains why the Site–Product-subset metrics are the fair way to compare the two graph formulations.

Third, the data contain strong short-term temporal structure. The final test period shows month-to-month variation, but the recent sales level remains highly informative. This property is important when interpreting the results, because simple short-memory baselines are expected to be competitive on such data.

5. Experimental Design

5.1. Aim of the experiments

The experiments in this thesis were designed to study a spectral–temporal graph neural network for industrial sales forecasting. The implemented architecture is based on the TGGC model, but the purpose of the experiments is broader than applying the original model directly. In this work, the TGGC architecture is used as a base spectral–temporal GNN framework in which both the graph operator and the graph spectral filter are treated as modelling choices. This terminology is important because the experiments compare several graph filters, not only the Gegenbauer filter used in the original TGGC formulation.

The main experimental question is whether the sales data can be represented usefully as a graph signal. The primary construction follows the idea proposed in this thesis: a fixed bipartite graph between customer groups and Site–Product nodes. This graph is meaningful for the application because it is based on observed customer–product interactions. As discussed in Chapter 4, a second formulation is also evaluated to serve as a denser baseline: a learned latent graph utilizing only the Site-Product nodes.

The experiments therefore compare two graph settings:

$$\text{fixed bipartite graph:} \qquad V = V_c \cup V_p,$$

and

$$\text{latent Site–Product graph:} \qquad V = V_p,$$

where V_c is the set of customer nodes and V_p is the set of Site–Product nodes. The fixed bipartite model is the main model proposed in the thesis, while the Site–Product-only model is used to test whether the sparse customer side of the graph helps, weakens, or approximately matches the forecasting performance on the Site–Product series.

5.2. Forecasting setup

The forecasting task is formulated as a supervised multi-step prediction problem. Let

$$X_t \in \mathbb{R}^N$$

be the graph signal in month t, where N is the number of nodes in the selected graph. For an input window of length W and a forecast horizon of length H, each training sample has the form

$$\mathbf{X}_{t-W+1:t} = (X_{t-W+1}, \ldots, X_t) \in \mathbb{R}^{W \times N},$$

with target

$$\mathbf{Y}_{t+1:t+H} = (X_{t+1}, \ldots, X_{t+H}) \in \mathbb{R}^{H \times N}.$$

In the final experiments,

$$W = 12, \qquad H = 3.$$

Thus, the model uses the previous twelve monthly observations and predicts the next three months.

The choice of a 12-month lookback window is motivated by the monthly structure of the data. One full year of history allows the models to observe possible annual demand patterns, seasonal effects, and repeated customer–product activity. At the same time, a longer input window would substantially reduce the number of available rolling training samples, which is problematic because the dataset contains only a limited number of monthly observations. The 3-month forecast horizon is chosen because it represents a short- to medium-term business planning horizon. It is more informative than a one-month-ahead forecast, while still being short enough to avoid an excessive reduction in the number of supervised learning windows.

The data are split chronologically into training, validation, and test parts. No random shuffling is used across time, because doing so would mix past and future observations and create an unrealistic forecasting setup. The split is defined by forecast origin and target months rather than by randomly selected rolling windows. This is important because rolling windows naturally overlap in their historical input periods. Such overlap is not considered leakage as long as each forecast uses only information available before its forecast origin. The relevant leakage risk would be using validation or test target values during training, normalization, hyperparameter tuning, or model selection.

For node i, the normalized signal is

$$\widetilde{x}_{t,i} = \frac{x_{t,i} - \mu_i}{\sigma_i},$$

where μ_i and σ_i are the mean and standard deviation of node i over the training period. All reported metrics are computed after transforming the predictions back to the original Net Sales scale.

The final reported hold-out test targets cover December 2025 to February 2026. A broader period may appear in the rolling-window construction because earlier months are required to form the input histories and forecast origins. However, the final reported test metrics are computed only on the final three target months. This test period is short, which is a limitation of the empirical design. It is nevertheless used as the final hold-out period because it represents the most recent available demand behaviour and therefore provides a realistic out-of-sample evaluation.

5.3. Graph formulations

5.3.1. Fixed bipartite graph

The fixed bipartite graph is the main graph construction of the thesis. The node set is split into customer nodes and Site–Product nodes:

$$V = V_c \cup V_p, \qquad V_c \cap V_p = \varnothing.$$

The weighted adjacency matrix has the block form

$$A = \begin{pmatrix} 0 & B \\ B^\top & 0 \end{pmatrix}.$$

The matrix B contains the customer–Site–Product interaction weights. If customer c purchased Site–Product p during the training period, then the corresponding entry B_{cp} is positive. In the experiments, these weights are based on historical absolute sales volume, with a logarithmic transformation applied to reduce the dominance of very large transactions.

The normalized graph Laplacian is

$$L = I - D^{-1/2} A D^{-1/2},$$

where D is the diagonal degree matrix. The corresponding graph shift used in most filters is

$$S = I - L = D^{-1/2} A D^{-1/2}.$$

For the fixed bipartite graph, this shift has a natural interpretation. One application of S moves information from one side of the graph to the other:

$$S \begin{pmatrix} x_c \\ x_p \end{pmatrix} = \begin{pmatrix} \widehat{B} x_p \\ \widehat{B}^\top x_c \end{pmatrix}.$$

Two applications return information to the same node type:

$$S^2 \begin{pmatrix} x_c \\ x_p \end{pmatrix} = \begin{pmatrix} \widehat{B}\widehat{B}^\top x_c \\ \widehat{B}^\top \widehat{B} x_p \end{pmatrix}.$$

Thus, odd powers of the graph shift describe customer–product propagation, while even powers describe induced similarity within customers or within Site–Product nodes.

5.3.2. Latent Site–Product graph

The second graph formulation uses only Site–Product nodes:

$$V = V_p.$$

In this model, the graph is not fixed from observed transactions. Instead, it is learned from the Site–Product time series by the latent-correlation mechanism in the TGGC architecture. This setting is closer to the original TGGC idea, where the graph structure is inferred from the temporal signals rather than supplied as a predefined adjacency matrix.

5.4. Model architecture

The implemented model is a TGGC-style spectral–temporal GNN. The architecture follows the compact pipeline

$$\text{Input} \rightarrow \text{Graph filtering} \rightarrow \text{Temporal filtering} \rightarrow \text{Forecast/backcast}.$$

This is illustrated schematically in Figure 5.1.

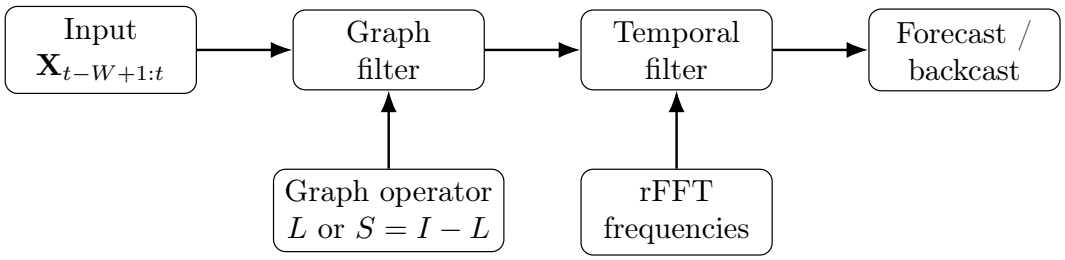

Figure 5.1.: Compact schematic of the TGGC-style spectral–temporal GNN architecture used in the experiments.

The graph spectral block applies a filter of the form

$$g_\Theta(S)X = \sum_{k=0}^{K-1} P_k(S)X\Theta_k,$$

where P_k is a graph filter basis and Θ_k is a learnable coefficient matrix. In the fixed bipartite experiment, S is obtained from the fixed graph. In the Site–Product experiment, the graph operator is obtained from the latent graph learned by the model.

The temporal block applies a frequency-domain transformation along the time dimension. The original TGGC model is motivated by spectral–temporal graph neural networks, which combine graph-domain filtering with temporal-domain filtering for multivariate time-series forecasting [33]. In this thesis, all available temporal Fourier modes are used rather than randomly selecting a subset, because the monthly input sequence is short. This avoids unnecessary stochastic variation in the temporal filter.

The final component maps the hidden representation to a forecast

$$\widehat{\mathbf{Y}}_{t+1:t+H} \in \mathbb{R}^{H\times N}.$$

A backcast output is also used during training, following the original TGGC-style implementation. The total training loss is

$$\mathcal{L} = \mathcal{L}_{\text{forecast}} + \lambda_{\text{back}}\mathcal{L}_{\text{backcast}},$$

where both terms are mean squared errors on normalized data and λ_{back} is a small backcast-loss weight.

5.5. Graph spectral filters

This section describes the graph spectral filters evaluated in the experiments. All filters are used within the same TGGC-style spectral–temporal architecture. The aim is therefore not to compare completely different forecasting architectures, but to isolate the effect of the graph spectral filter while keeping the remaining forecasting setup fixed.

Let $G = (V, E, A)$ be a weighted undirected graph with $N = |V|$ nodes, adjacency matrix A, degree matrix D, and normalized graph Laplacian

$$L = I - D^{-1/2} A D^{-1/2}.$$

The normalized graph shift used in most filters is defined as

$$S = I - L.$$

For an undirected graph, L is symmetric and positive semidefinite, and its eigenvalues lie in $[0, 2]$. Consequently, the eigenvalues of $S = I - L$ lie in $[-1, 1]$. This makes S suitable for polynomial bases that are naturally defined on bounded intervals, such as Chebyshev, Gegenbauer, Jacobi, and Legendre polynomials.

A graph spectral filtering layer maps an input graph signal

$$X \in \mathbb{R}^{N \times d_{\text{in}}}$$

to an output representation

$$Z \in \mathbb{R}^{N \times d_{\text{out}}}.$$

The general polynomial form used in this thesis is

$$Z = g_\Theta(S)X = \sum_{k=0}^{K-1} P_k(S) X \Theta_k,$$

where K is the filter order, $P_k(S)$ is the k-th graph filter basis operator, and

$$\Theta_k \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$$

is a trainable weight matrix. The different filter families correspond to different choices of P_k. In the spectral domain, if $S = U \Lambda_S U^\top$, then applying $P_k(S)$ corresponds to applying $P_k(\lambda)$ to each graph frequency λ. Therefore, the filter basis determines how graph frequencies are amplified, suppressed, or combined.

Monomial filter

The monomial filter is the simplest polynomial graph filter. It uses powers of the graph shift:

$$P_k(S) = S^k.$$

The resulting graph filter is

$$g_\Theta^{\text{mono}}(S)X = \sum_{k=0}^{K-1} S^k X \Theta_k.$$

Here, $S^0 = I$ preserves the original node signal, S^1 propagates information over one graph step, S^2 propagates information over two graph steps, and so on. Thus, the monomial filter has a direct spatial interpretation as a weighted combination of multi-hop graph propagation.

In the fixed bipartite graph, this interpretation is especially important. Odd powers of S transfer information between customer nodes and Site–Product nodes, while even powers return information to nodes of the same type through shared neighbours. Therefore, the monomial filter provides a simple baseline for testing whether multi-hop propagation on the fixed graph is useful for forecasting.

Taylor-type filter

The Taylor-type filter uses scaled powers of the graph shift:

$$P_k(S) = \frac{S^k}{k!}.$$

The corresponding filter is

$$g_\Theta^{\text{Taylor}}(S)X = \sum_{k=0}^{K-1} \frac{S^k}{k!} X \Theta_k.$$

The factorial term reduces the magnitude of higher-order powers. Compared with the plain monomial filter, this gives less weight to very high-order propagation terms before training. It can therefore be interpreted as a smoother version of the monomial basis. This filter is included to test whether scaled multi-hop propagation behaves more stably than unscaled powers of S.

GCN-type first-order filter

The GCN-type filter is included as a simple first-order graph convolution baseline. It is motivated by the first-order approximation of spectral graph convolution used in the standard GCN model [23]. In the notation of this thesis, the basis is

$$P_0(S) = I, \qquad P_1(S) = S,$$

with no higher-order terms. The filter is therefore

$$g_\Theta^{\text{GCN}}(S)X = X\Theta_0 + SX\Theta_1.$$

This filter only uses the original node signal and one-step graph propagation. It tests whether local neighbourhood smoothing is sufficient, compared with higher-order polynomial and rational spectral filters.

Chebyshev filter

The Chebyshev filter uses Chebyshev polynomials of the first kind. These are defined by

$$T_0(x) = 1, \qquad T_1(x) = x,$$

and the recurrence

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x), \qquad k \geq 2.$$

The graph filter is

$$g_\Theta^{\text{Cheb}}(S)X = \sum_{k=0}^{K-1} T_k(S)X\Theta_k.$$

Chebyshev polynomial filters are widely used in spectral graph convolution because they provide an efficient approximation of spectral filters without computing the full eigendecomposition of the graph Laplacian [28]. They are also localized, since a polynomial of order $K-1$ only uses information up to $K-1$ graph steps.

The Chebyshev recurrence is designed for inputs whose spectrum lies in $[-1, 1]$. For this reason, using the correctly scaled graph operator is important. If the operator is not properly scaled, or if the graph has a difficult spectral structure, the recurrence can lead to numerical instability. This is relevant for the empirical results in this thesis, where the Chebyshev filter behaves differently on the fixed bipartite graph and the learned latent Site–Product graph.

Gegenbauer filter

The Gegenbauer filter is central to the original TGGC architecture [33]. Gegenbauer polynomials $C_k^{(\alpha)}(x)$ are a family of orthogonal polynomials on $[-1, 1]$, controlled by a shape parameter

$$\alpha > -\frac{1}{2}.$$

They are orthogonal with respect to the weight function

$$w_\alpha(x) = (1 - x^2)^{\alpha - \frac{1}{2}}.$$

The first two Gegenbauer polynomials are

$$C_0^{(\alpha)}(x) = 1, \qquad C_1^{(\alpha)}(x) = 2\alpha x.$$

For $k \geq 2$, the recurrence can be written as

$$C_k^{(\alpha)}(x) = \frac{1}{k}\left[2x(k + \alpha - 1)C_{k-1}^{(\alpha)}(x) - (k + 2\alpha - 2)C_{k-2}^{(\alpha)}(x)\right].$$

The corresponding graph filter is

$$g_\Theta^{\text{Gegen}}(S)X = \sum_{k=0}^{K-1} C_k^{(\alpha)}(S)X\Theta_k.$$

Here, α determines the particular Gegenbauer basis, while Θ_k are the trainable model parameters. In the experiments, α is treated as a filter hyperparameter or fixed according to the implemented filter setting, whereas Θ_k are learned during training.

The Gegenbauer basis is included because it is the defining graph filter of TGGC. The motivation is that an appropriate orthogonal polynomial basis can provide a more expressive and better-conditioned spectral approximation than simple monomial powers.

Jacobi filter

The Jacobi filter uses Jacobi polynomials $P_k^{(a,b)}(x)$, which form a broad family of orthogonal polynomials on $[-1, 1]$. The parameters

$$a > -1, \qquad b > -1$$

control the weight function

$$w_{a,b}(x) = (1 - x)^a (1 + x)^b.$$

The graph filter is

$$g_\Theta^{\text{Jacobi}}(S)X = \sum_{k=0}^{K-1} P_k^{(a,b)}(S) X \Theta_k.$$

Here, a and b define the Jacobi polynomial family, while Θ_k are trainable weight matrices. The Jacobi family is more general than several other classical polynomial families. For example, Legendre polynomials correspond to the special case $a = b = 0$, and Gegenbauer polynomials can also be related to a special symmetric Jacobi family.

The Jacobi filter is included to test whether the additional flexibility of two shape parameters improves forecasting performance compared with the more restricted Gegenbauer and Legendre bases.

Legendre filter

The Legendre filter uses Legendre polynomials $P_k(x)$, which are orthogonal on $[-1, 1]$ with respect to the constant weight function. They are a special case of Jacobi polynomials with $a = b = 0$. The first two Legendre polynomials are

$$P_0(x) = 1, \qquad P_1(x) = x,$$

and the recurrence is

$$(k + 1)P_{k+1}(x) = (2k + 1)xP_k(x) - kP_{k-1}(x).$$

The graph filter is

$$g_\Theta^{\text{Legendre}}(S)X = \sum_{k=0}^{K-1} P_k(S) X \Theta_k.$$

The Legendre filter is included as a simple orthogonal polynomial basis on the same spectral interval as the graph shift S. Compared with Gegenbauer and Jacobi filters, it does not introduce additional shape parameters.

Hermite filter

The Hermite filter uses Hermite polynomials $H_k(x)$. In the physicists' convention, these are defined by

$$H_0(x) = 1, \qquad H_1(x) = 2x,$$

and

$$H_{k+1}(x) = 2xH_k(x) - 2kH_{k-1}(x).$$

The graph filter is

$$g_\Theta^{\text{Hermite}}(S)X = \sum_{k=0}^{K-1} H_k(S)X\Theta_k.$$

Hermite polynomials are classically orthogonal on the real line with respect to a Gaussian weight, rather than specifically on $[-1, 1]$. In this thesis, they are evaluated on the bounded graph shift S and are included as an additional polynomial basis for empirical comparison. Their inclusion should therefore be interpreted as a filter-family comparison, not as a claim that the graph spectrum has a Gaussian Hermite weight structure.

Bernstein filter

The Bernstein filter is applied to the rescaled Laplacian

$$X_L = \frac{L}{2}.$$

This scaling is used because the eigenvalues of L lie in $[0, 2]$, so the eigenvalues of X_L lie in $[0, 1]$. Bernstein polynomials are naturally defined on this interval.

The Bernstein basis polynomial of degree K is

$$B_{k,K}(x) = \binom{K}{k} x^k (1-x)^{K-k}, \qquad k = 0, \ldots, K.$$

The graph filter is

$$g_\Theta^{\text{Bern}}(X_L)X = \sum_{k=0}^{K} B_{k,K}(X_L)X\Theta_k.$$

Here, K is the Bernstein degree, $B_{k,K}$ is the k-th Bernstein basis polynomial, and Θ_k is the corresponding trainable weight matrix.

Bernstein filters have been used in graph neural networks because they provide a stable basis for approximating flexible spectral responses on $[0, 1]$ [30]. Unlike the monomial basis, the Bernstein basis is nonnegative on $[0, 1]$ and the basis functions sum to one. This can improve numerical behaviour when approximating graph spectral responses.

Laguerre filter

The Laguerre filter is also applied to the rescaled Laplacian

$$X_L = \frac{L}{2},$$

because Laguerre polynomials are naturally associated with nonnegative domains. The first two Laguerre polynomials are

$$\mathcal{L}_0(x) = 1, \qquad \mathcal{L}_1(x) = 1 - x,$$

and the recurrence is

$$(k+1)\mathcal{L}_{k+1}(x) = (2k + 1 - x)\mathcal{L}_k(x) - k\mathcal{L}_{k-1}(x).$$

The graph filter is

$$g_\Theta^{\text{Laguerre}}(X_L)X = \sum_{k=0}^{K-1} \mathcal{L}_k(X_L)X\Theta_k.$$

The Laguerre filter is included as another structured polynomial basis. Since it is not naturally tied to the interval $[-1, 1]$, it is applied to the nonnegative rescaled Laplacian rather than to the shifted operator S.

ARMA filter

The ARMA filter is a rational graph filter. Unlike polynomial filters, which use only finite powers of a graph operator, rational filters contain inverse terms. This allows them to represent longer-range or diffusion-like graph responses with a relatively small filter order.

In this thesis, the ARMA-type basis operators are written as

$$P_k(S) = (I - \alpha S)^{-1}S^k, \qquad k = 0, \dots, K-1.$$

The full filter is therefore

$$g_\Theta^{\text{ARMA}}(S)X = \sum_{k=0}^{K-1} (I - \alpha S)^{-1}S^k X\Theta_k.$$

The parameter $\alpha \in \mathbb{R}$ controls the strength of the rational diffusion term. For stability, α must be chosen so that $I - \alpha S$ is invertible. A sufficient condition is

$$|\alpha|\rho(S) < 1,$$

where $\rho(S)$ is the spectral radius of S. Under this condition, the inverse can be expanded as a Neumann series:

$$(I - \alpha S)^{-1} = \sum_{j=0}^{\infty} \alpha^j S^j.$$

This identity explains the long-range effect of the ARMA filter. Even if the explicit filter order K is small, the inverse term implicitly contains infinitely many powers of the graph shift, with weights controlled by α. The trainable matrices Θ_k then determine how these rationally filtered components are combined.

ARMA graph filters are motivated by autoregressive moving-average ideas in the graph spectral domain and have been used to design graph neural networks with rational spectral responses [31]. In the context of this thesis, the ARMA filter is especially relevant for the fixed bipartite graph, where useful information may need to travel through several alternating customer–Site–Product paths.

Cayley filter

The Cayley filter is another rational spectral filter. It is based on a Cayley transform of the graph operator. In the implementation used in this thesis, the Cayley-type operator is written as

$$C_h(S) = (I - hS)^{-1}(I + hS),$$

where $h > 0$ is a scaling parameter. The basis operators are

$$P_k(S) = C_h(S)^k, \qquad k = 0, \dots, K-1.$$

The graph filter is therefore

$$g_\Theta^{\text{Cayley}}(S)X = \sum_{k=0}^{K-1} C_h(S)^k X \Theta_k.$$

The parameter h controls the spectral transformation induced by the Cayley operator. As with ARMA, the inverse term requires $I - hS$ to be invertible. In the spectral domain, if $Su = \lambda u$, then

$$C_h(S)u = \frac{1 + h\lambda}{1 - h\lambda} u.$$

Thus, the Cayley transform maps each graph frequency λ to the rationally transformed value

$$\lambda \mapsto \frac{1 + h\lambda}{1 - h\lambda}.$$

The Cayley filter therefore does not simply combine ordinary graph-walk powers. It first transforms the graph spectrum using a rational map and then applies powers of the transformed operator. This can produce frequency-selective behaviour that differs from low-order polynomial filters.

Cayley spectral filters were introduced to allow more flexible graph frequency responses than standard polynomial graph convolution [32]. In this thesis, the Cayley filter is included to test whether such rational spectral transformations are useful for demand forecasting on the learned latent Site–Product graph and the fixed bipartite graph.

Summary of filter families

Table 5.1 summarizes the filter families used in the experiments after the individual definitions above.

Table 5.1.: Spectral filter families evaluated in the experiments.

Filter	Class	Operator domain	Basis operator
Monomial	Polynomial	$S \in [-1, 1]$	$P_k(S) = S^k$
Taylor-type	Polynomial	$S \in [-1, 1]$	$P_k(S) = S^k/k!$
GCN-type	First-order polynomial	$S \in [-1, 1]$	$P_0(S) = I,\ P_1(S) = S$
Chebyshev	Orthogonal polynomial	$S \in [-1, 1]$	$P_k(S) = T_k(S)$
Gegenbauer	Orthogonal polynomial	$S \in [-1, 1]$	$P_k(S) = C_k^{(\alpha)}(S)$
Jacobi	Orthogonal polynomial	$S \in [-1, 1]$	$P_k(S) = P_k^{(a,b)}(S)$
Legendre	Orthogonal polynomial	$S \in [-1, 1]$	$P_k(S) = P_k(S)$
Hermite	Orthogonal polynomial	$S \in [-1, 1]$	$P_k(S) = H_k(S)$
Bernstein	Polynomial	$X_L = L/2 \in [0, 1]$	$P_k(X_L) = B_{k,K}(X_L)$
Laguerre	Polynomial	$X_L = L/2 \in [0, 1]$	$P_k(X_L) = \mathcal{L}_k(X_L)$
ARMA	Rational	$S \in [-1, 1]$	$P_k(S) = (I - \alpha S)^{-1} S^k$
Cayley	Rational	$S \in [-1, 1]$	$P_k(S) = \left[(I - hS)^{-1}(I + hS)\right]^k$

The table highlights the main distinction between the filter families. Polynomial filters approximate graph spectral responses using finite polynomial bases. They are localized and have a direct multi-hop propagation interpretation. Rational filters, such as ARMA and Cayley, contain inverse graph operators. They can therefore represent more global or frequency-selective graph responses, but they require more careful numerical treatment. This distinction is important for the experiments because the best-performing filter differs between the fixed bipartite graph and the learned latent Site–Product graph.

5.6. Temporal filtering

The temporal filtering component is applied after the graph spectral filtering step. Let

$$X_{t-W+1:t} \in \mathbb{R}^{N \times W}$$

denote one input window, where N is the number of nodes and W is the lookback length. In the experiments, each node is represented by one monthly demand sequence. Hence, the observed input contains one scalar value per node and month, without additional external covariates.

The graph spectral filtering step maps this input window to an internal node-level representation

$$H_G \in \mathbb{R}^{N \times W \times d_h},$$

where d_h denotes the number of learned hidden channels. These channels are model-internal representations and are not observed input features.

For each node $i \in \{1, \dots, N\}$ and hidden channel $r \in \{1, \dots, d_h\}$, the temporal signal is the sequence

$$h_{i,r} = \big(H_G(i, 1, r), \dots, H_G(i, W, r)\big) \in \mathbb{R}^W.$$

The temporal spectral module applies a real Fourier transform along the time dimension:

$$\widehat{h}_{i,r} = \mathcal{F}_t(h_{i,r}) \in \mathbb{C}^{\lfloor W/2 \rfloor + 1}.$$

Equivalently, for the full tensor,

$$\widehat{H}_G = \mathcal{F}_t(H_G),$$

where $\mathcal{F}_t$ acts only on the temporal index.

A learnable frequency-domain transformation is then applied to the Fourier coefficients. In abstract form, this operation is written as

$$\widehat{H}_T = \Phi_\Psi(\widehat{H}_G),$$

where Φ_Ψ denotes the temporal spectral filter and Ψ its learnable parameters. The filtered representation is transformed back to the time domain by the inverse Fourier transform:

$$H_T = \mathcal{F}_t^{-1}\left(\widehat{H}_T\right) = \mathcal{F}_t^{-1}\left(\Phi_\Psi\left(\mathcal{F}_t(H_G)\right)\right).$$

Thus, the temporal block can be summarized as

$$H_G \xrightarrow{\mathcal{F}_t} \widehat{H}_G \xrightarrow{\Phi_\Psi} \widehat{H}_T \xrightarrow{\mathcal{F}_t^{-1}} H_T.$$

The purpose of this block is to model temporal dependencies through frequency components rather than directly in the time domain. Since the input window consists of twelve monthly observations, all available Fourier modes are retained in the implementation. This avoids discarding potentially relevant frequency information in a short monthly sequence and keeps the temporal component fixed across the graph-filter comparisons.

Combining the graph and temporal spectral parts, one TGGC-style spectral–temporal block can be written abstractly as

$$H_G = g_\Theta(S) X_{t-W+1:t} = \sum_{k=0}^{K-1} P_k(S) X_{t-W+1:t} \Theta_k,$$

followed by

$$H_T = \mathcal{F}_t^{-1}\left(\Phi_\Psi\left(\mathcal{F}_t(H_G)\right)\right).$$

Here, S denotes the graph shift operator, $P_k(S)$ is the k-th graph filter basis, Θ_k are the graph-filter parameters, and Φ_Ψ is the learnable temporal frequency filter. The forecast and backcast heads are then applied to H_T.

5.7. Baseline models

Several baseline models are included to make the graph neural results interpretable. The baselines are divided into traditional forecasting methods, deep temporal baselines, and graph-based models.

5.7.1. Traditional forecasting baselines

The simplest baselines are the naive last-value forecast, the seasonal naive forecast, and moving averages. The naive forecast uses

$$\widehat{x}_{t+h} = x_t,$$

while the moving average with window m uses

$$\widehat{x}_{t+h} = \frac{1}{m} \sum_{j=0}^{m-1} x_{t-j}.$$

These baselines are important because sales series often have strong persistence. A graph neural model should only be considered useful if it is compared against such short-memory methods.

A ridge autoregression baseline is also included. Ridge regression solves

$$\min_{\beta} \left\{ \|Y - X\beta\|_2^2 + \lambda \|\beta\|_2^2 \right\},$$

where λ controls the strength of the L^2 regularization. This follows the classical ridge regression idea of stabilizing least-squares estimation with a quadratic penalty [12]. A vector autoregression baseline is included to model linear dependence between multiple series:

$$Y_t = A_1 Y_{t-1} + \cdots + A_p Y_{t-p} + u_t,$$

which is the standard VAR formulation for multivariate time series [11]. An ARIMA baseline is also evaluated as a classical univariate time-series method, following the autoregressive integrated moving-average framework [5].

5.7.2. Deep temporal baselines

The deep learning baselines are LSTM, GRU, and Temporal CNN. The LSTM was introduced to address the difficulty of learning long-term dependencies in recurrent neural networks by using gated memory cells [13]. The GRU is a related gated recurrent architecture introduced in the RNN encoder–decoder framework [14]. Both are standard sequence models and are included as recurrent neural baselines.

The Temporal CNN baseline uses one-dimensional temporal convolutions. Temporal convolutional networks have been shown to be strong sequence-modelling baselines and can be competitive with recurrent architectures on many tasks [15]. In this thesis, the Temporal CNN baseline tests whether local convolution over time is sufficient without using an explicit graph structure.

5.8. Training procedure and evaluation

All neural models are trained using the training split and monitored on the validation split. Early stopping is applied according to the validation loss, and the best validation

model is restored before test evaluation.To account for stochastic variation in neural network training, all neural models are evaluated over five random seeds. The reported neural results are therefore given as mean and standard deviation across seeds. This is important because random weight initialization, mini-batch ordering, and stochastic optimization can affect the final model performance. Deterministic baselines such as moving averages, naive forecasts, ARIMA, and ridge autoregression are repeated under the same evaluation protocol and therefore have zero standard deviation across runs. The loss is computed on normalized data, while MAE, RMSE, and WAPE are reported on the original sales scale.

Hyperparameter tuning is performed using the validation period. The main tuned choices include the model family, graph spectral filter, learning rate, hidden dimension, dropout, number of training epochs, and early-stopping behaviour. The tuning procedure is intentionally limited to a controlled set of configurations. A very large hyperparameter search would be difficult to justify because the monthly dataset provides only a small number of effective training windows and could lead to overfitting on the validation period. The purpose of the experiments is therefore to compare model families and graph-filter choices under a consistent evaluation protocol rather than to claim that each model is exhaustively optimized.

For the fixed bipartite model, two types of metrics are computed:

$$\text{full bipartite metrics on } V_c \cup V_p,$$

and

$$\text{Site–Product subset metrics on } V_p.$$

This distinction is necessary because the full bipartite graph contains sparse customer nodes, while the Site–Product-only model predicts only V_p. Therefore, the fair comparison between the fixed bipartite model and the Site–Product model is made on the common Site–Product node set.

The main metrics are

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \widehat{y}_i|,$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2},$$

and

$$\text{WAPE} = \frac{\sum_{i=1}^{n} |y_i - \widehat{y}_i|}{\sum_{i=1}^{n} |y_i|}.$$

WAPE is used as the primary comparison metric because it normalizes total absolute error by the total absolute demand. This is useful when comparing errors across multiple nodes with different sales volumes.

5.9. Experimental comparisons

The experiments are organized around four comparisons.

First, the fixed bipartite TGGC is compared across spectral filters. This tests which graph filter is most suitable for the customer–Site–Product graph.

Second, the latent Site–Product TGGC is compared across the same spectral filters. This tests whether the same filter remains best when the graph is learned only from Site–Product time series.

Third, the graph neural models are compared with traditional baselines and deep temporal baselines. This comparison is necessary because a graph model is only useful if it provides additional value beyond simple temporal persistence and generic sequence modelling.

Fourth, the fixed bipartite model is evaluated on the Site–Product subset and compared with the Site–Product-only model. This is the key comparison for the thesis. It tests whether using customer information through a fixed bipartite graph improves, weakens, or approximately matches the latent graph learned directly between Site–Product nodes.

The results of these experiments are presented in Chapter 6.

6. Results

6.1. Experimental overview and interpretation strategy

This chapter reports the experimental results after repeating the stochastic neural-network experiments with five random seeds. The results are therefore reported as mean ± standard deviation wherever a model is affected by random initialization. Deterministic baselines, such as the moving average and naive forecasts, have zero standard deviation because they are identical across repeated runs.

The chapter is organized around the three research questions:

RQ1 Are TGGC-style graph models competitive with classical and neural baselines for monthly Site–Product demand forecasting?

RQ2 Does the fixed bipartite customer–Site–Product graph improve forecasting compared with a latent Site–Product graph?

RQ3 How does the choice of spectral graph filter affect forecasting performance and numerical robustness?

The experimental design uses a 12-month input window and a 3-month forecasting horizon. The 12-month lookback is motivated by the monthly frequency of the data: it gives the model one full annual cycle while keeping the number of trainable windows as large as possible. The 3-month horizon corresponds to a short quarterly planning horizon. However, this design also creates an important limitation. The available monthly history is short, so after using a 12-month lookback and a 3-month horizon, the number of effective supervised windows is limited. The test targets cover the period from September 2025 to February 2026, generated from four rolling test origins. Hence, the test period is informative but not long enough to support strong statistical claims about all model rankings.

Two graph formulations are evaluated. The first is the fixed bipartite graph proposed in this thesis, where customer nodes are connected to Site–Product nodes by observed customer–product interactions. The second is the latent Site–Product graph, where the graph is learned directly among Site–Product series. For a fair comparison, the fixed bipartite model is evaluated both on the full graph and on the common Site–Product subset. The Site–Product subset is the main quantity for comparing the two graph formulations.

6.1.1. Evaluation metrics

WAPE is used as the primary evaluation metric because it is directly interpretable relative to total demand volume. In this thesis, WAPE measures the total absolute forecast error divided by the total absolute actual demand over the evaluation period. This makes it suitable for a business forecasting setting where the practical question is how large the aggregate forecasting error is relative to observed sales volume.

RMSE is additionally reported as a secondary metric. It measures the error magnitude in the original target scale, Net Sales K, and penalizes large errors more strongly than MAE or WAPE. This is useful as a robustness check because a model with acceptable aggregate WAPE may still produce large misses on individual high-volume Site–Product series. However, RMSE is scale-dependent and can be dominated by high-volume nodes. Therefore, WAPE remains the main model-selection criterion, while RMSE is used to check whether the same conclusions hold when large absolute errors are weighted more strongly.

6.2. RQ1: Are TGGC-style models competitive with baselines?

Table 6.1 summarizes the central finding. The best overall method on the Site–Product targets is not a neural graph model, but the 3-month moving average. It obtains a WAPE of 27.47% and an RMSE of 227.40. The naive last-value forecast is very close, with 27.67% WAPE and 231.81 RMSE. This shows that the dataset contains strong short-term temporal persistence.

Table 6.1.: Main results on the common Site–Product target nodes. Neural models are reported over five seeds. WAPE is the primary metric; RMSE is reported as a secondary robustness metric in the original Net Sales K scale.

Family	Model	Filter	Runs	SP WAPE	SP RMSE	SP MAE	Parameters
Traditional baseline	Moving average 3	–	5	27.47 ± 0.00	227.40 ± 0.00	121.44 ± 0.00	0
Traditional baseline	Naive last value	–	5	27.67 ± 0.00	231.81 ± 0.00	122.34 ± 0.00	0
Fixed bipartite TGGC	Fixed Bipartite TGGC (arma)	arma	5	35.73 ± 3.30	273.67 ± 18.75	157.94 ± 14.58	3 348 318
Traditional baseline	Moving average 6	–	5	35.93 ± 0.00	309.15 ± 0.00	158.87 ± 0.00	0
Traditional baseline	ARIMA(1 0 0)	–	5	38.57 ± 0.00	297.31 ± 0.00	170.50 ± 0.00	0
Traditional baseline	Ridge autoregression	–	5	39.34 ± 0.00	293.60 ± 0.00	173.91 ± 0.00	0
Deep baseline	LSTM	–	5	50.25 ± 0.66	363.08 ± 4.69	222.14 ± 2.91	4 579
Deep baseline	GRU	–	5	50.15 ± 0.34	362.40 ± 2.55	221.69 ± 1.49	3 459
Deep baseline	Temporal CNN	–	5	52.03 ± 1.58	376.13 ± 9.89	230.05 ± 6.97	3 331
Fixed bipartite TGGC	Fixed Bipartite TGGC (cheby)	cheby	5	192.74 ± 204.46	2094.13 ± 2540.77	852.10 ± 903.95	3 348 318

The fixed bipartite TGGC with the ARMA filter is the best graph model in the fixed bipartite experiment, with a Site–Product WAPE of 35.73 ± 3.30% and an RMSE of 273.67 ± 18.75. This is better than the LSTM, GRU, and Temporal CNN baselines in the fixed bipartite experiment, but it is still worse than the 3-month moving average on both WAPE and RMSE. Therefore, the empirical results show that the graph models are useful for studying graph construction and spectral filters, but they do not provide the best predictive accuracy in this dataset.

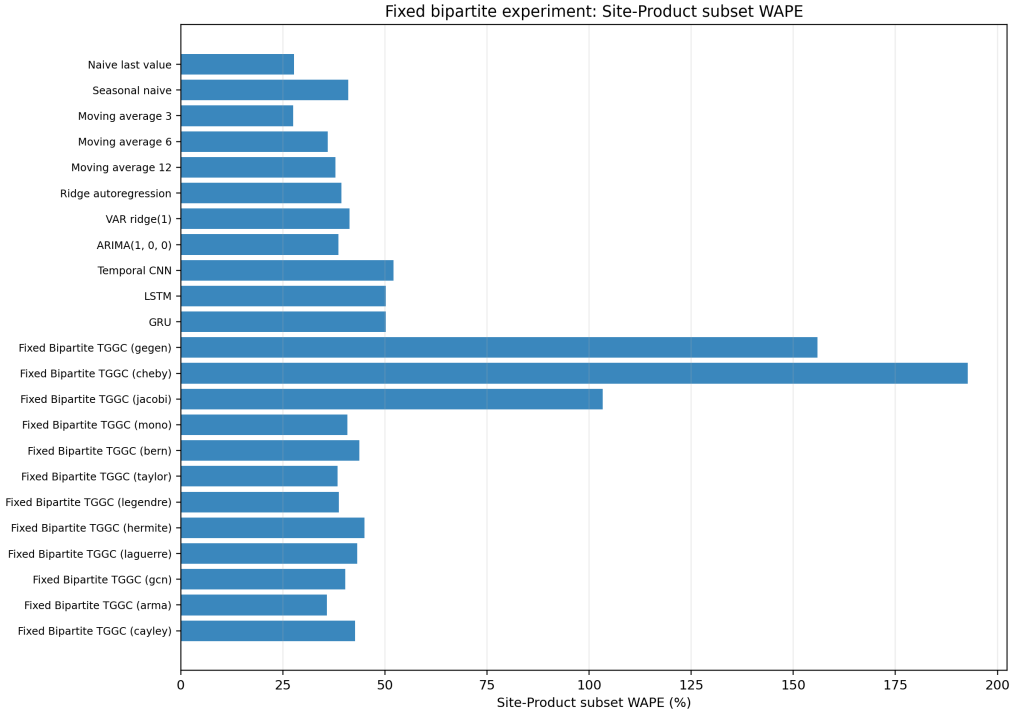

Figure 6.1.: Updated fixed bipartite experiment: Site–Product-subset WAPE by model family.

The result has an important implication for the thesis claim. The evidence does not support the statement that explicit graph structure improves forecasting accuracy over simple baselines. Instead, it supports the weaker and more accurate statement that explicit graph structure provides an interpretable modelling framework and can be competitive with learned graph models, while simple temporal persistence remains the strongest predictor in the current data. The RMSE results strengthen this conclusion because the moving-average baseline also has a lower absolute error magnitude than the best graph-based models.

6.3. RQ2: Fixed bipartite graph versus latent Site–Product graph

The fixed bipartite graph and the latent Site–Product graph answer different modelling questions. The fixed graph is interpretable because it directly represents customer–product relations. The latent graph is less directly interpretable, but it can adapt its connections to the forecasting loss.

Table 6.2 compares the best graph model from each formulation on the common Site–Product target nodes. The latent Site–Product TGGC with the Cayley filter obtains $34.75 \pm 3.16\%$ WAPE and 269.64 ± 21.50 RMSE. The fixed bipartite TGGC with the

ARMA filter obtains 35.73 ± 3.30% WAPE and 273.67 ± 18.75 RMSE. The fixed graph is therefore slightly worse than the learned Site–Product graph, but the difference is small compared with the gap to the moving-average baseline.

Table 6.2.: Direct comparison of graph formulations on the common Site–Product target nodes. WAPE is reported in percent; RMSE and MAE are in the original Net Sales K scale.

Model	Best filter	Runs	WAPE	RMSE	MAE
Moving average 3	–	5	27.47 ± 0.00%	227.40 ± 0.00	121.44 ± 0.00
Fixed bipartite TGGC	ARMA	5	35.73 ± 3.30%	273.67 ± 18.75	157.94 ± 14.58
Latent Site–Product TGGC	Cayley	5	34.75 ± 3.16%	269.64 ± 21.50	153.63 ± 13.96

Compared with the 3-month moving-average baseline, the best fixed bipartite TGGC model remains worse by approximately 8.26 WAPE percentage points, while the best latent Site–Product TGGC model remains worse by approximately 7.28 WAPE percentage points. This confirms that neither graph formulation improves the best predictive accuracy. However, the fixed bipartite graph remains close to the learned graph while preserving a more transparent economic interpretation: it keeps customers and Site–Product nodes as separate node types and uses observed interactions as edges.

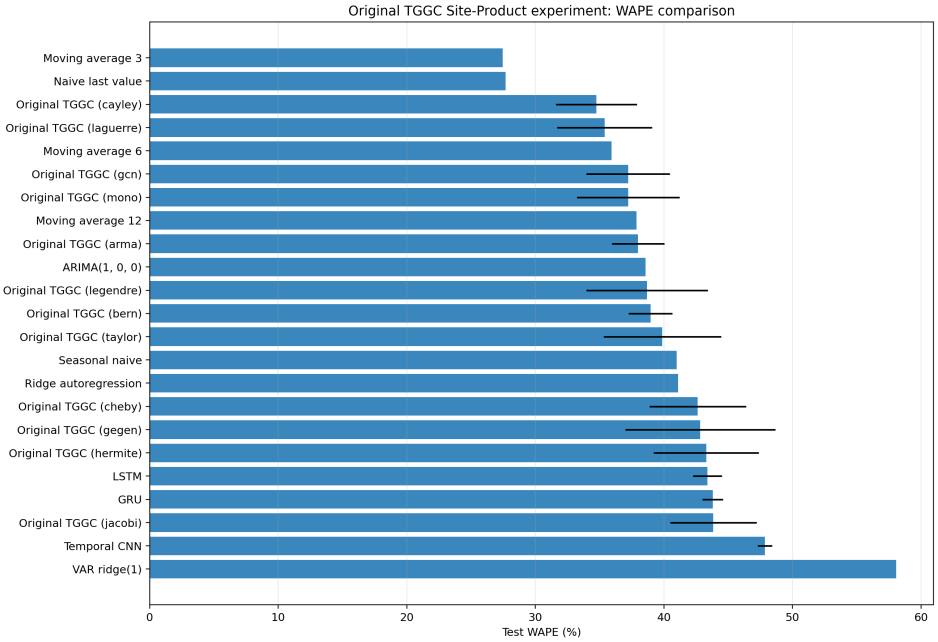

Figure 6.2.: Updated Site–Product-only experiment: WAPE comparison across baselines and TGGC variants.

6.4. RQ3: Effect of spectral filter choice

The spectral filter has a strong effect on performance and stability. Table 6.3 compares filter-wise WAPE and RMSE on the common Site–Product targets for both graph formulations. The best filter changes with the graph operator: ARMA is best for the fixed bipartite graph, while Cayley is best for the latent Site–Product graph. RMSE gives the same broad conclusion: the rational ARMA and Cayley variants are among the most stable and accurate graph filters, whereas the Chebyshev filter is highly unstable on the fixed bipartite graph.

Table 6.3.: Updated filter-wise comparison on Site–Product targets. Values are mean ± standard deviation over five seeds. WAPE is reported in percent; RMSE is in the original Net Sales K scale.

Filter	Fixed WAPE	Fixed RMSE	Latent WAPE	Latent RMSE
arma	35.73 ± 3.30	273.67 ± 18.75	37.99 ± 2.03	288.79 ± 11.41
cayley	42.61 ± 3.70	314.83 ± 21.27	34.75 ± 3.16	269.64 ± 21.50
laguerre	43.16 ± 5.53	325.18 ± 31.55	35.38 ± 3.70	272.55 ± 21.99
gcn	40.28 ± 2.77	305.20 ± 13.78	37.21 ± 3.26	285.67 ± 17.62
mono	40.80 ± 5.96	306.37 ± 37.43	37.22 ± 3.99	286.03 ± 25.09
taylor	38.33 ± 3.12	294.74 ± 17.74	39.87 ± 4.57	301.38 ± 27.18
legendre	38.74 ± 4.91	294.15 ± 28.29	38.68 ± 4.73	292.59 ± 30.35
bern	43.72 ± 5.43	326.46 ± 29.14	38.95 ± 1.70	294.02 ± 13.76
hermite	44.97 ± 3.45	330.31 ± 22.29	43.28 ± 4.09	321.83 ± 26.97
jacobi	103.27 ± 87.18	1095.69 ± 1055.49	43.84 ± 3.36	327.05 ± 24.91
gegen	155.95 ± 236.66	1336.18 ± 2173.93	42.83 ± 5.83	320.19 ± 40.30
cheby	192.74 ± 204.46	2094.13 ± 2540.77	42.63 ± 3.75	321.45 ± 19.09

The ARMA and Cayley filters are both rational filters. This is relevant because rational filters can represent frequency responses that are difficult to approximate with low-order polynomial filters. In the fixed bipartite graph, ARMA may be helpful because information alternates between customer nodes and Site–Product nodes. A rational diffusion-like response can aggregate information over multiple alternating paths without requiring a very high polynomial order. In the latent Site–Product graph, Cayley performs best, suggesting that the learned graph operator benefits from a different rational transformation.

6.4.1. Instability of selected orthogonal polynomial filters

The fixed bipartite graph reveals a clear instability for selected orthogonal polynomial filters. This effect is not limited to the Chebyshev filter. On the latent Site–Product graph, Chebyshev, Jacobi, and Gegenbauer are not among the strongest filters, but they remain numerically ordinary: Chebyshev obtains $42.63 \pm 3.75\%$ WAPE, Jacobi obtains $43.84 \pm 3.36\%$, and Gegenbauer obtains $42.83 \pm 5.83\%$. Their RMSE values are also close to the range of the other weaker filters.

On the fixed bipartite graph, however, the same filter families behave very differently. Jacobi obtains $103.27 \pm 87.18\%$ WAPE and 1095.69 ± 1055.49 RMSE, Gegenbauer obtains $155.95 \pm 236.66\%$ WAPE and 1336.18 ± 2173.93 RMSE, and Chebyshev obtains

$192.74 \pm 204.46\%$ WAPE and 2094.13 ± 2540.77 RMSE. The large standard deviations indicate that these filters are not merely weaker on average, but also highly sensitive across random seeds.

This behaviour suggests that the instability is related to the interaction between the fixed bipartite graph operator and these orthogonal polynomial bases. A plausible explanation is the sparse two-block structure of the bipartite graph. In this graph, customer nodes and Site–Product nodes form two different node types, and polynomial powers of the graph shift propagate information through alternating paths. Odd powers transfer information across node types, while even powers return information to the same node type through shared neighbours. If the graph is sparse and the available monthly series are short, these repeated propagation paths may lead to high-variance aggregation and sensitivity to initialization.

A second explanation is spectral-density mismatch. Orthogonal polynomial filters are theoretically attractive when their weight functions are well matched to the graph signal density. This condition may hold for some benchmark graphs, but it is not guaranteed for the fixed customer–Site–Product graph used in this thesis. The graph is an economic interaction graph constructed from observed transactions, not a physical sensor graph or a dense similarity graph. Therefore, the spectral distribution of the fixed bipartite operator may be poorly matched to the weight functions associated with Chebyshev, Jacobi, and Gegenbauer bases.

These explanations should be interpreted as plausible mechanisms rather than causal proof. The experiments do not separately isolate sparsity, graph spectrum, learning rate, normalization, polynomial order, and parameter count. Nevertheless, the contrast between the fixed bipartite graph and the latent Site–Product graph shows that the stability of orthogonal polynomial filters depends strongly on the graph operator. In this setting, the rational filters are more robust: ARMA performs best on the fixed bipartite graph, while Cayley performs best on the latent Site–Product graph.

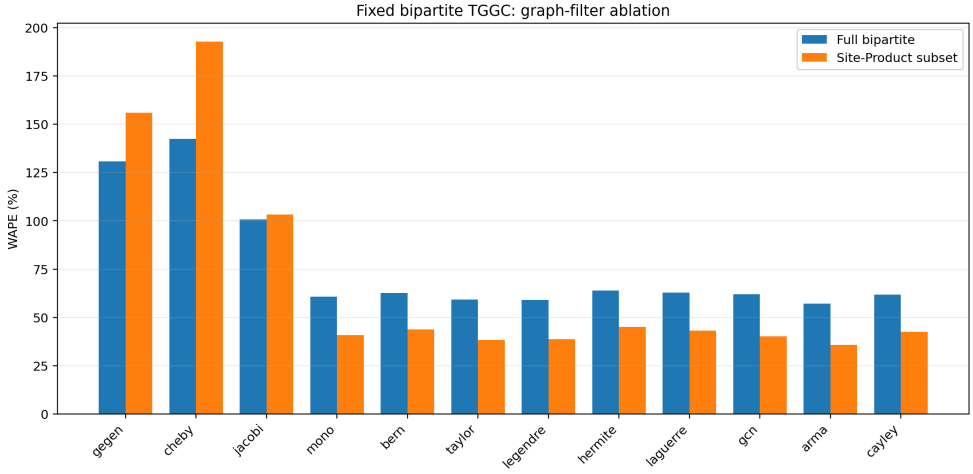

Figure 6.3.: Updated filter ablation for the fixed bipartite graph. Jacobi, Gegenbauer, and Chebyshev should be interpreted as unstable selected orthogonal polynomial filters in this graph formulation.

‘‘‘

6.5. Error analysis and robustness

6.5.1. Horizon-wise error

The best fixed bipartite TGGC model is evaluated over a 3-step forecast horizon. Table 6.4 shows that the Site–Product WAPE increases from 28.76% at horizon 1 to 35.25% at horizon 3. Site–Product RMSE also increases from 228.12 to 280.32 over the same horizon. This pattern is expected because each additional step moves further away from the observed 12-month input window.

Table 6.4.: Horizon-wise error for the best fixed bipartite TGGC model. RMSE and MAE are in the original Net Sales K scale.

Horizon	SP WAPE	Full WAPE	SP RMSE	SP MAE
1	28.76%	50.68%	228.12	130.35
2	30.55%	53.17%	239.80	132.38
3	35.25%	57.41%	280.32	154.99

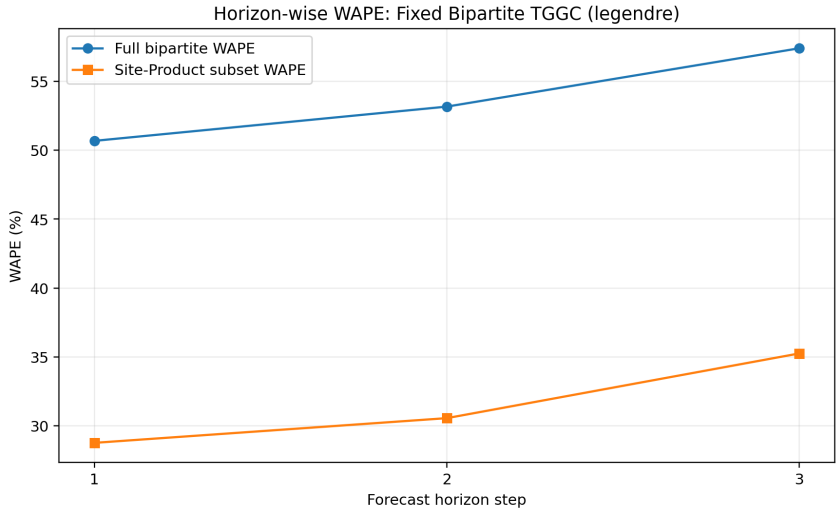

Figure 6.4.: Horizon-wise WAPE for the best fixed bipartite TGGC model.

6.5.2. Node type, sparsity, and WAPE concentration

The fixed bipartite graph contains many sparse customer nodes. Table 6.5 shows that customer nodes have a much higher aggregate WAPE than Site–Product nodes. This does not mean that the absolute customer-node errors are always large; rather, many customer series have small or zero denominators, which makes relative percentage errors unstable. RMSE gives the complementary view: Site–Product nodes have much larger absolute error magnitudes because their sales volumes are larger, while customer-node WAPE is higher because many customer denominators are small.

Table 6.5.: Node-type error for the best fixed bipartite TGGC model. RMSE and MAE are in the original Net Sales K scale.

Node type	MAE	RMSE	WAPE
customer	9.94	30.58	75.32%
site_product	139.24	250.41	31.49%

The sparsity of the test period is substantial. In the Site–Product-only node set, 7 out of 28 nodes have zero total actual sales in the evaluated test targets. In the fixed bipartite evaluation, 501 out of 965 customer nodes have zero total actual sales, while 6 Site–Product nodes have zero total actual sales. Therefore, node-level percentage errors must be interpreted carefully. Aggregate WAPE is more stable because it sums absolute errors and absolute actuals before division. RMSE does not suffer from a zero denominator, but it is still strongly affected by the absolute scale of each node and therefore emphasizes high-volume series.

The Site–Product demand is also volume-concentrated. The top 3 Site–Product nodes account for approximately 38.9% of total absolute test demand, the top 5 account for 57.7%, and the top 10 account for 88.3%. Consequently, aggregate WAPE is influenced strongly by high-volume nodes. This is appropriate for a business-weighted forecasting metric, but it also means that improvements on low-volume intermittent nodes may not visibly change the total WAPE. RMSE is even more sensitive to high-volume nodes and large individual misses, so it should be interpreted as a complementary error-magnitude metric rather than as a replacement for WAPE.

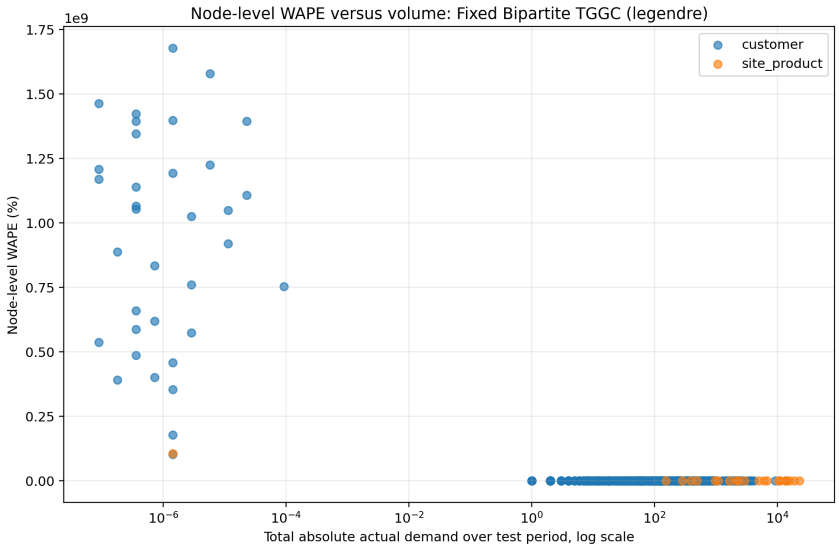

Figure 6.5.: Node volume versus WAPE for the fixed bipartite model. Low-volume nodes can have unstable percentage errors.

Table 6.6.: Compact summary of the research-question answers.

RQ	Question	Answer from the experiments
RQ1	Are TGGC-style graph models competitive with classical and neural baselines for monthly Site–Product demand forecasting?	The TGGC-style graph models are competitive with the tested deep temporal baselines, but they do not outperform the strongest classical baseline. The 3-month moving average achieves the best overall Site–Product WAPE of 27.47%, while the best graph model reaches 34.75–35.73% WAPE depending on the graph formulation. This indicates that short-term temporal persistence is the strongest signal in the dataset.
RQ2	Does the fixed bipartite customer–Site–Product graph improve forecasting compared with a latent Site–Product graph?	The fixed bipartite graph does not improve forecasting accuracy over the latent Site–Product graph. The best fixed bipartite model, using an ARMA filter, achieves $35.73 \pm 3.30\%$ WAPE, while the best latent Site–Product model, using a Cayley filter, achieves $34.75 \pm 3.16\%$ WAPE. However, the fixed graph remains close to the latent graph and has the advantage of a more interpretable customer–product structure.
RQ3	How does the choice of spectral graph filter affect forecasting performance and numerical robustness?	The spectral filter choice has a substantial effect on both accuracy and stability. ARMA performs best on the fixed bipartite graph, while Cayley performs best on the latent Site–Product graph. Rational filters are generally more stable and accurate than several polynomial alternatives in the experiments, whereas the Jacobi, Gegenbauer, and Chebyshev, become unstable on the fixed bipartite graph.

6.5.3. Summary of findings

Overall, the 3-month moving average remains the strongest model according to both WAPE and RMSE. The best graph-based models are approximately 7–8 WAPE percentage points behind this baseline and also have higher RMSE. The fixed bipartite graph is slightly worse than the learned latent Site–Product graph, but the difference between the two graph formulations is small relative to the gap to the moving average. The filter choice matters substantially: rational filters such as ARMA and Cayley behave better than several polynomial alternatives, while selected orthogonal polynomial filters, especially Jacobi, Gegenbauer, and Chebyshev, are unstable on the fixed bipartite graph. Therefore, the results do not show an accuracy gain from graph structure, but they do show that graph construction and spectral filter choice materially affect forecasting behaviour, robustness, and interpretability.

7. Conclusion

This thesis investigated whether explicit customer–product graph structure improves monthly industrial demand forecasting. The main modelling idea was to represent the data as a fixed bipartite graph between customer nodes and Site–Product nodes, and to apply TGGC-style spectral–temporal graph neural networks with different graph spectral filters. A latent Site–Product graph was also evaluated in order to separate the effect of the proposed fixed graph from the effect of using a graph neural architecture in general.

The experiments were repeated over five random seeds. The best forecasting method on the common Site–Product target nodes is not a neural graph model, but the simple 3-month moving average. The naive last-value forecast is almost equally strong. This shows that the dominant predictive signal in the dataset is short-term temporal persistence and the sales data contain a strong local temporal structure that should not be ignored. For the available monthly data, recent sales observations explain a large part of the forecastable variation.

The graph neural models therefore do not improve predictive accuracy over the strongest simple baselines. This is the central empirical result of the thesis. The best fixed bipartite TGGC model uses an ARMA filter and the best latent Site–Product TGGC model uses a Cayley filter. Both graph models are clearly worse than the 3-month moving average. Thus, although the graph-based models capture meaningful structural information, they do not outperform the simple temporal persistence baseline in predictive accuracy.

Nevertheless, the graph-based experiments are still informative. First, the fixed bipartite graph performs close to the learned Site–Product graph. Its WAPE is only about 2.81% worse than the latent graph in relative terms. This suggests that the explicit customer–Site–Product structure contains useful information, even though it is not sufficient to outperform simple temporal persistence. Second, the fixed graph is more interpretable than the latent graph, because its edges correspond to observed customer–product relations rather than learned hidden similarity. Third, the experiments show that graph construction and spectral filter choice interact strongly: the best filter is ARMA for the fixed bipartite graph, while it is Cayley for the latent Site–Product graph.

The spectral-filter results are an important mathematical finding of the thesis. Rational filters perform best in the two strongest graph models. This suggests that rational spectral transformations may be more suitable than low-order polynomial filters for these data and graph operators. The ARMA filter may be useful on the fixed bipartite graph because information alternates between customer nodes and Site–Product nodes, so useful signals may require propagation through multiple alternating paths. The Cayley filter, by contrast, performs best on the learned Site–Product graph, where the graph operator represents latent similarity among target series rather than an explicit two-mode economic relation.

The instability of selected orthogonal polynomial filters requires a separate interpretation. On the fixed bipartite graph, it obtains a Site–Product WAPE of 192.74 ± 204.46%, which is not only poor but also highly unstable across seeds. This indicates that Chebyshev is not merely a weak filter in this setting; it behaves as a numerically unstable outlier. Possible causes include spectral scaling, the sparse two-block structure of the bipartite graph, repeated polynomial propagation through alternating node types, learning-rate sensitivity, and the high parameter count of the fixed bipartite model relative to the limited number of monthly training windows.

Overall, the thesis does not demonstrate that graph neural networks outperform simple baselines for this monthly industrial demand forecasting problem. Instead, its contribution is more specific: it shows how explicit bipartite customer–product structure can be formulated for spectral–temporal graph forecasting, how this structure compares with a learned Site–Product graph, and how different spectral filters behave under these graph constructions. The negative accuracy result is valuable because it shows that complex graph neural architectures should be benchmarked against simple persistence-based methods, especially when the available time series are short, sparse, and dominated by recent demand behaviour.

Limitations

Several limitations must be considered when interpreting the results.

First, the available time series is short for training neural forecasting models. The monthly data cover the period from January 2023 to February 2026. With a 12-month input window and a 3-month forecast horizon, the number of effective supervised training windows is limited. This is a serious constraint for neural architectures, especially for the fixed bipartite TGGC model, which has more than three million trainable parameters. The strong performance of the 3-month moving average suggests that the data may not contain enough repeated temporal patterns for the more complex models to learn stable nonlinear structure beyond short-term persistence.

Second, the test period is relatively short. The evaluation covers test targets from September 2025 to February 2026, generated from four rolling test origins. This is useful for comparing models under a consistent setup, but it is not long enough to support strong statistical conclusions about all model rankings. The repeated five-seed evaluation improves robustness for stochastic neural models, but it does not remove the limitation caused by a short historical dataset and a short test period.

Third, the dataset is sparse and intermittent, especially on the customer side of the fixed bipartite graph. Many customer nodes have zero or very low sales in the evaluation period. This makes percentage-based error measures difficult to interpret at node level, because small denominators can create very large relative errors. For this reason, the Site–Product-subset WAPE is the most relevant metric for comparing the fixed bipartite graph with the latent Site–Product graph.

Fourth, aggregate WAPE is dominated by high-volume Site–Product nodes. In the updated analysis, the top 3 Site–Product nodes account for approximately 38.9% of

total absolute test demand, the top 5 for 57.7%, and the top 10 for 88.3%. This makes WAPE appropriate from a business-weighted accuracy perspective, but it also means that improvements on low-volume or intermittent nodes may be hidden in the aggregate metric. A model could improve forecasts for sparse nodes without substantially changing total WAPE.

Fifth, the fixed bipartite graph may be more interpretable than predictive. Its edges represent observed customer–product relations, which is useful for explaining the structure of the business data. However, the experiments do not show that these edges add enough predictive information to beat short-memory baselines. The graph may therefore be more valuable as a structural representation and diagnostic tool than as a direct source of forecast accuracy in the current setup.

Sixth, the neural graph models may be overparameterized for the available monthly data. This is especially relevant for the fixed bipartite TGGC, where the number of nodes and parameters is much larger than in the latent Site–Product-only model. With few training windows, the model can become sensitive to initialization, early stopping, and hyperparameter choices. The instability of Jacobi, Gegenbauer, and Chebyshev is an extreme example of this issue, but the broader concern also applies to other filters and architectures.

Seventh, the experiments use only historical sales values and graph structure. Important exogenous variables are not included. In industrial demand forecasting, sales may depend on prices, customer contracts, promotions, supply limitations, stock availability, holidays, inflation, macroeconomic conditions, and known business events. Without such covariates, the models must infer all changes from past sales alone, which limits their ability to forecast structural shifts or exceptional months.

Eighth, the hyperparameter tuning is necessarily limited. The experiments compare many spectral filters and model classes, but a full search over all architecture sizes, learning rates, regularization strengths, graph normalizations, and filter orders would be computationally expensive. The reported results therefore compare reasonable experimental configurations rather than globally optimal versions of each model.

Finally, the results are based on one company-specific sales dataset. The conclusions are reliable for the studied data and forecasting setup, but they should not be generalized to all industrial demand forecasting problems without further evidence. Other datasets with longer histories, denser customer–product interactions, stronger seasonality, or richer covariates may lead to different results.

Future work

The updated results suggest several directions for future work.

A first direction is to reduce model complexity. Since the moving average outperforms all neural architectures, a simpler graph model may be more appropriate than a highly parameterized spectral–temporal network. Possible alternatives include graph-regularized linear models, low-rank graph models, shallow graph convolutional models, or residual models that forecast only the deviation from a moving-average baseline.

A second direction is to combine temporal persistence with graph structure more directly. Instead of asking the graph neural network to learn the full forecast, the model could first compute a strong persistence baseline and then learn graph-based residual corrections. This would test whether the graph contains information beyond recent sales behaviour.

A third direction is to improve the bipartite graph construction. The current graph is fixed and based on observed customer–product interactions. Future work could use time-varying edge weights, edge weights based on recent purchase intensity, customer clustering, product hierarchy information, or graph pruning to reduce sparsity and noise.

A fourth direction is to include exogenous covariates. Prices, promotions, customer-specific events, supply constraints, holidays, and macroeconomic indicators may explain demand changes that are not visible from historical sales alone. Including these variables could make neural models more useful and reduce their dependence on short-memory persistence.

A fifth direction is to study filter stability more systematically. The instability of Jacobi, Gegenbauer, and Chebyshev should be tested under different spectral scalings, graph normalizations, learning rates, gradient clipping strategies, and polynomial orders. This would help distinguish whether the instability is mainly caused by implementation choices, optimization, the spectrum of the fixed bipartite graph, or the interaction between all of these factors.

A final direction is to evaluate the models on longer and more diverse datasets. More years of monthly data, additional business units, or external benchmark datasets would make it possible to test whether the conclusions of this thesis are specific to the current data or reflect a more general limitation of complex graph neural networks for sparse monthly demand forecasting.

Bibliography

[1] Jonathan D. Cryer. *Time Series Analysis*. Duxbury Press, 1986.

[2] Jongseon Kim, Hyungjoon Kim, HyunGi Kim, Dongjun Lee, and Sungroh Yoon. A comprehensive survey of time series forecasting: Architectural diversity and open challenges, 2024.

[3] Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: A survey. *Philosophical Transactions of the Royal Society A*, 379(2194):20200209, 2021.

[4] Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Yuyang Wang, Danielle Maddix, Caner Turkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella, Laurent Callot, and Tim Januschowski. Deep learning for time series forecasting: Tutorial and literature survey. *ACM Computing Surveys*, 55(6):1–36, 2022.

[5] George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung. *Time Series Analysis: Forecasting and Control*. Wiley, 5 edition, 2015.

[6] Rob J. Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 3 edition, 2021.

[7] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(9):11121–11128, 2023.

[8] Robert G. Brown. Statistical forecasting for inventory control. Technical report, Arthur D. Little, 1959.

[9] Charles C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *Office of Naval Research Memorandum*, 1957.

[10] Peter R. Winters. Forecasting sales by exponentially weighted moving averages. *Management Science*, 6(3):324–342, 1960.

[11] Helmut Lütkepohl. *New Introduction to Multiple Time Series Analysis*. Springer, 2005.

[12] Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[14] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734, 2014.

[15] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[17] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[18] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11106–11115, 2021.

[19] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

[20] Shizhan Liu, Hang Yu, Cong Liao, Jianguo Li, Weiyao Lin, Alex X. Liu, and Schahram Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *International Conference on Learning Representations*, 2022.

[21] Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. In *Proceedings of the 39th International Conference on Machine Learning*, 2022.

[22] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[23] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[24] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3634–3640, 2018. arXiv:1709.04875.

[25] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[26] Yao Ma, Ziyi Guo, Zhaochun Ren, Jiliang Tang, and Dawei Yin. Streaming graph neural networks. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 719–728, 2020.

[27] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. In *ICML Workshop on Graph Representation Learning*, 2020.

[28] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

[29] Gábor Szegő. *Orthogonal Polynomials*. American Mathematical Society, 1939.

[30] Mingguo He, Zhewei Wei, Zengfeng Huang, and Hongteng Xu. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

[31] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3496–3507, 2022.

[32] Ron Levie, Federico Monti, Xavier Bresson, and Michael M. Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109, 2019.

[33] Ming Jin, Guangsi Shi, Yuan-Fang Li, Bo Xiong, Tian Zhou, Flora D. Salim, Liang Zhao, Lingfei Wu, Qingsong Wen, and Shirui Pan. Towards expressive spectral-temporal graph neural networks for time series forecasting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(6):4926–4940, 2025.

[34] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1263–1272, 2017.

[35] Davide Bacciu, Federico Errica, Alessio Micheli, and Marco Podda. A gentle introduction to deep learning for graphs. *Neural Networks*, 129:203–221, 2020.

[36] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[37] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[38] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[39] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020.

[40] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021.

[41] Fan R. K. Chung. *Spectral Graph Theory*, volume 92 of *CBMS Regional Conference Series in Mathematics*. American Mathematical Society, 1997.

[42] David I. Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.

A. Appendix

This appendix reports the essential implementation components used in the empirical experiments. The complete code was implemented in Python using PyTorch. Only the main parts required to reproduce the forecasting setup, graph construction, evaluation metrics, and training protocol are shown.

A.1. Rolling-window construction and chronological split

The forecasting task is constructed with a fixed lookback window of twelve months and a forecast horizon of three months. The split is chronological and is defined over rolling-window samples.

```
def make_windows(data, window_size, horizon):
    X, y, labels = [], [], []
    T = data.shape[0]

    for i in range(T - window_size - horizon + 1):
        X.append(data[i:i + window_size])
        y.append(data[i + window_size:i + window_size + horizon])
        labels.append(months[i + window_size:i + window_size + horizon])

    return (
        np.asarray(X, dtype=np.float32),
        np.asarray(y, dtype=np.float32),
        np.asarray(labels)
    )


X_raw, y_raw, target_months = make_windows(
    raw_matrix,
    WINDOW_SIZE,
    HORIZON
)

num_samples, _, num_nodes = X_raw.shape

train_end = int(TRAIN_RATIO * num_samples)
val_end = int((TRAIN_RATIO + VAL_RATIO) * num_samples)

idx_train = np.arange(0, train_end)
idx_val = np.arange(train_end, val_end)
idx_test = np.arange(val_end, num_samples)
```

Listing A.1: Rolling-window construction for supervised forecasting samples.

A.2. Training-only normalization

To avoid leakage, normalization parameters are estimated only from the training-visible period. The same mean and standard deviation are then applied to validation and test windows.

```
train_time_cutoff = idx_train[-1] + WINDOW_SIZE + HORIZON
train_period_raw = raw_matrix[:train_time_cutoff]

train_mean = train_period_raw.mean(axis=0)
train_std = train_period_raw.std(axis=0)
train_std = np.where(train_std < 1e-8, 1.0, train_std)

X_norm = (
    X_raw - train_mean.reshape(1, 1, -1)
) / train_std.reshape(1, 1, -1)

y_norm = (
    y_raw - train_mean.reshape(1, 1, -1)
) / train_std.reshape(1, 1, -1)


def denormalize_np(arr_norm):
    return (
        arr_norm * train_std.reshape(1, 1, -1)
        + train_mean.reshape(1, 1, -1)
    )
```

Listing A.2: Training-only normalization to avoid leakage.

A.3. Dataset and data loaders

The PyTorch dataset stores the normalized input and target windows together with their sample indices. The training loader is shuffled using a seed-specific generator, while validation and test loaders are evaluated chronologically.

```
class WindowDataset(Dataset):
    def __init__(self, X, y, indices):
        self.X = torch.tensor(X[indices], dtype=torch.float32)
        self.y = torch.tensor(y[indices], dtype=torch.float32)
        self.indices = np.asarray(indices)

    def __len__(self):
        return len(self.indices)

    def __getitem__(self, i):
        return self.X[i], self.y[i], int(self.indices[i])


def make_loaders(seed):
    generator = torch.Generator()
    generator.manual_seed(seed)

    train_loader = DataLoader(
        WindowDataset(X_norm, y_norm, idx_train),
        batch_size=BATCH_SIZE,
        shuffle=True,
```

```
        generator=generator
    )

    val_loader = DataLoader(
        WindowDataset(X_norm, y_norm, idx_val),
        batch_size=BATCH_SIZE,
        shuffle=False
    )

    test_loader = DataLoader(
        WindowDataset(X_norm, y_norm, idx_test),
        batch_size=BATCH_SIZE,
        shuffle=False
    )

    return train_loader, val_loader, test_loader
```

Listing A.3: Window dataset and seed-controlled data loaders.

A.4. Forecast evaluation metrics

```
def metrics_np(y_true, y_pred, eps=1e-8):
    err = y_pred - y_true

    mae = float(np.mean(np.abs(err)))
    rmse = float(np.sqrt(np.mean(err ** 2)))

    wape = float(
        np.sum(np.abs(err))
        / (np.sum(np.abs(y_true)) + eps)
    )

    return {
        "MAE": mae,
        "RMSE": rmse,
        "WAPE_%": 100.0 * wape
    }


def split_metrics_np(true_raw, pred_raw):
    full = metrics_np(true_raw, pred_raw)

    sp = metrics_np(
        true_raw[:, :, SITE_PRODUCT_INDICES],
        pred_raw[:, :, SITE_PRODUCT_INDICES]
    )

    customer = metrics_np(
        true_raw[:, :, CUSTOMER_INDICES],
        pred_raw[:, :, CUSTOMER_INDICES]
    )

    return full, sp, customer
```

Listing A.4: Evaluation metrics used in the experiments.

A.5. Fixed bipartite graph construction

The fixed bipartite graph is constructed from customer–Site–Product interactions. To avoid leakage, the graph can be restricted to the training-visible history only. Customers and Site–Product combinations are represented as separate node types, and observed sales interactions define the weighted bipartite edges.

```
# Use only training-visible months for graph construction.
if EDGE_WEIGHT_FROM_TRAIN_ONLY:
    graph_months = set(months[:train_time_cutoff])
    graph_df = df[df["month"].astype(str).isin(graph_months)].copy()
else:
    graph_df = df.copy()

# Edge weights are based on observed sales magnitudes.
if USE_ABS_SALES_FOR_GRAPH:
    graph_df["edge_weight_value"] = np.abs(graph_df["sales"])
else:
    graph_df["edge_weight_value"] = np.maximum(graph_df["sales"], 0.0)

edge_table = (
    graph_df
    .groupby([CUSTOMER_COL, SITE_PRODUCT_COL])["edge_weight_value"]
    .sum()
    .reset_index()
)

num_customer_nodes = len(customer_nodes)
num_site_product_nodes = len(site_product_nodes)
num_nodes = num_customer_nodes + num_site_product_nodes

A = np.zeros((num_nodes, num_nodes), dtype=np.float32)

for _, row in edge_table.iterrows():
    c_idx = customer_to_idx[row[CUSTOMER_COL]]
    sp_idx = num_customer_nodes + site_product_to_idx[row[SITE_PRODUCT_COL]]
    weight = row["edge_weight_value"]

    A[c_idx, sp_idx] = weight
    A[sp_idx, c_idx] = weight

# Symmetric normalized Laplacian: L = I - D^{-1/2} A D^{-1/2}
degree = A.sum(axis=1)
degree_safe = np.where(degree > 0, degree, 1.0)
D_inv_sqrt = np.diag(1.0 / np.sqrt(degree_safe))

A_norm = D_inv_sqrt @ A @ D_inv_sqrt
L = np.eye(num_nodes, dtype=np.float32) - A_norm

fixed_adjacency = torch.tensor(A, dtype=torch.float32)
fixed_laplacian = torch.tensor(L, dtype=torch.float32)
```

Listing A.5: Construction of the fixed customer–Site–Product bipartite graph.

A.6. Fixed bipartite TGGC model

In the fixed graph variant, the latent graph construction of the original TGGC model is replaced by a fixed normalized bipartite graph Laplacian. The Laplacian and adjacency matrix are registered as non-trainable buffers.

```
class FixedBipartiteTGGC(OriginalTGGCModel):
    """
    TGGC architecture with the learned latent graph replaced
    by a fixed customer--Site--Product bipartite graph.
    """

    def __init__(self, fixed_laplacian, fixed_adjacency, *args, **kwargs):
        super().__init__(*args, **kwargs)

        self.register_buffer(
            "fixed_laplacian",
            fixed_laplacian.float()
        )

        self.register_buffer(
            "fixed_adjacency",
            fixed_adjacency.float()
        )

    def _eye(self, laplacian):
        return torch.eye(
            laplacian.size(0),
            device=laplacian.device,
            dtype=torch.float32
        )

    def _graph_shift(self, laplacian):
        # S = I - L
        return self._eye(laplacian) - laplacian

    def _unit_laplacian(self, laplacian):
        # X = L / 2, used for filters on [0,1]
        return 0.5 * laplacian
```

Listing A.6: Fixed bipartite TGGC model using a precomputed graph Laplacian.

A.7. Graph spectral filter bases

Different graph filters are implemented by generating filter bases from the graph Laplacian. The experiments compare polynomial and rational filter families, including Chebyshev, ARMA, and Cayley filters.

```
def Mono_polynomial(self, laplacian):
    # Monomial basis: I, S, S^2, ..., S^{K-1}
    S = self._graph_shift(laplacian)
    basis = [self._eye(laplcian)]

    for _ in range(1, self.order):
        basis.append(torch.matmul(basis[-1], S))

    return torch.stack(basis, dim=0)
```

```
10
11
12 def Taylor_polynomial(self, laplacian):
13     # Taylor-like basis: S^k / k!
14     S = self._graph_shift(laplacian)
15     basis = []
16
17     for k in range(self.order):
18         basis.append(
19             self._mat_power(S, k) / float(math.factorial(k))
20         )
21
22     return torch.stack(basis, dim=0)
23
24
25 def GCN_polynomial(self, laplacian):
26     # First-order GCN-like basis: [I, S, 0, ..., 0]
27     I = self._eye(laplacian)
28     S = self._graph_shift(laplacian)
29     Z = torch.zeros_like(I)
30
31     basis = [I]
32
33     if self.order > 1:
34         basis.append(S)
35
36     while len(basis) < self.order:
37         basis.append(Z)
38
39     return torch.stack(basis[:self.order], dim=0)
```

Listing A.7: Examples of graph spectral filter bases.

A.8. Model evaluation

```
1  def evaluate_torch(model, loader):
2      model.eval()
3
4      preds, trues, sample_idxs = [], [], []
5      mse_sum, nb = 0.0, 0
6      mse = nn.MSELoss()
7
8      with torch.no_grad():
9          for xb, yb, sidx in loader:
10             xb = xb.to(DEVICE)
11             yb = yb.to(DEVICE)
12
13             out = model(xb)
14             pred = out[0] if isinstance(out, tuple) else out
15
16             loss = mse(pred, yb)
17             mse_sum += float(loss.item())
18             nb += 1
19
20             preds.append(pred.detach().cpu().numpy())
21             trues.append(yb.detach().cpu().numpy())
22             sample_idxs.extend(sidx.numpy().tolist())
23
```

```
    pred_norm = np.concatenate(preds, axis=0)
    true_norm = np.concatenate(trues, axis=0)

    pred_raw = denormalize_np(pred_norm)
    true_raw = denormalize_np(true_norm)

    full, sp, customer = split_metrics_np(true_raw, pred_raw)

    full["MSE_norm"] = mse_sum / max(nb, 1)
    full["RMSE_norm"] = math.sqrt(full["MSE_norm"])

    return full, sp, customer, pred_raw, true_raw, np.asarray(sample_idxs)
```

Listing A.8: Evaluation of neural forecasting models.