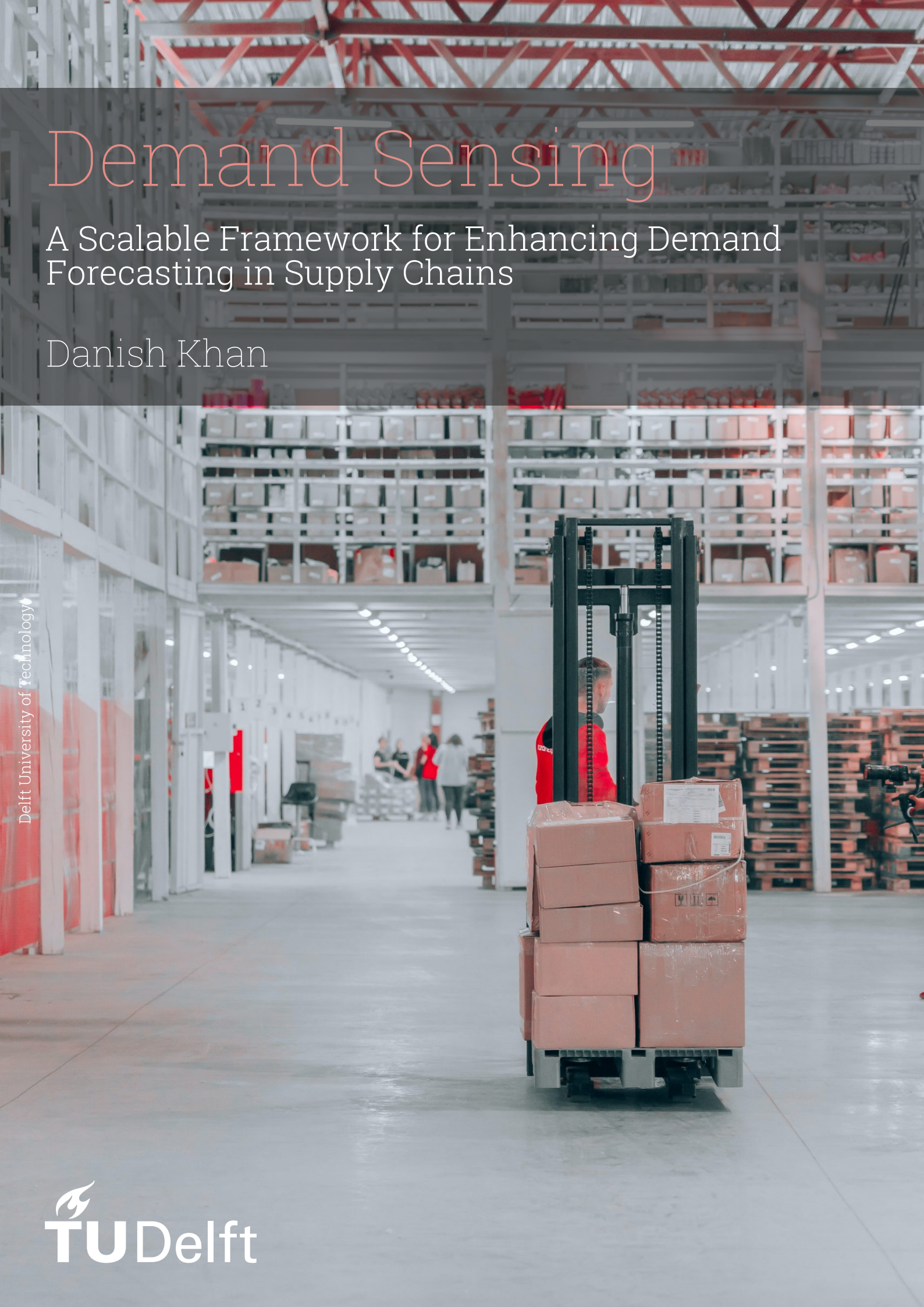
Demand Sensing

A Scalable Framework for Enhancing Demand Forecasting in Supply Chains

Danish Khan



Delft University of Technology

TU Delft

Demand Sensing

A Scalable Framework for Enhancing Demand Forecasting in Supply Chains

by

Danish Khan

Supervisor: Prof. dr. Fabian Mies
Master Program: Applied Mathematics - Mathematics of Data Science
Faculty: Faculty of Electrical Engineering, Mathematics & Computer Science (EEMCS), Delft

TU Delft

Abstract

This thesis develops a mathematical and computational framework for demand sensing in supply chain management, addressing the dual challenges of forecast accuracy and hierarchical coherence. This work was carried out during a graduate internship at Dassault Systèmes. Modern retail demand is shaped by vertical aggregation constraints, horizontal product interactions, and temporal dependencies influenced by external drivers. Traditional approaches (classical time series, machine learning, and reconciliation heuristics) treat these aspects separately and struggle to deliver coherent and accurate forecasts at scale.

The thesis first establishes a theoretical foundation for multi-product demand forecasting. Forecasting is formalized as a supervised learning problem on probability spaces, introducing the multi-product demand process and a general forecast operator that unifies local versus global modeling, recursive versus direct horizons, and alternative loss functions. Hierarchical forecasting is expressed in compact linear algebraic form, showing how classical heuristics and optimal reconciliation (MinT/OLS) can be understood as projections onto the coherent subspace.

Building on this, the thesis introduces a graph-based extension of the framework. By embedding time series into relational graphs, Graph Neural Networks (GNNs) capture vertical, horizontal, and temporal dependencies within a unified model. An end-to-end forecasting pipeline is proposed, combining graph construction, feature extraction with external covariates, GNN encoding, and forecast generation, with coherence ensured through bottom-up construction, regularization penalties, or post-hoc reconciliation.

Empirical evaluation on the M5 dataset demonstrates the practical implications. Incorporating external drivers consistently improves accuracy across models and aggregation levels. Machine learning models, particularly gradient boosting, outperform classical baselines at granular levels, though the latter remain competitive at higher aggregations. The proposed Hierarchical Graph Network (HGN) achieves competitive results, with particular benefits at intermediate levels where both hierarchical and cross-series relations are most informative. A comparison of local versus global training highlights the trade-off between accuracy and computational efficiency.

The findings underscore three insights for practice: external drivers require robust data infrastructure, hybrid local–global modeling offers balanced accuracy and efficiency, and reconciliation remains essential to guarantee coherence. Overall, the thesis demonstrates how integrating hierarchical structure, external signals, and graph-based learning advances both the theory and practice of demand sensing in modern supply chains.

Contents

1 Introduction

1.1. Background

Forecasting plays a central role in supply chain management, underpinning decisions on inventory control, production planning, and distribution scheduling. Accurate forecasts help reduce safety stock, prevent stockouts, and improve customer service levels. However, as supply chains have become larger, faster, and more digital, demand patterns have become increasingly complex, volatile, and interconnected [27, 12]. Traditional forecasting approaches, designed for stable, low-dimensional environments, struggle to cope with this complexity.

Recent studies highlight three forms of dependencies that are central to modern demand forecasting:

- *Vertical dependencies:* Aggregation structures (e.g., Stock-Keeping-Units (SKUs) $\rightarrow$ category $\rightarrow$ region $\rightarrow$ total) impose coherence constraints: forecasts at lower levels must sum to forecasts at higher levels. Violating these constraints causes planning mismatches between operational and strategic decisions [11].
- *Horizontal dependencies:* Products within the same level often move together, driven by shared market forces, promotions, or cannibalization effects. This cross-series correlation is a major source of information that univariate models ignore [4, 13].
- *Temporal dependencies:* Sales data exhibits seasonality, trends, and autocorrelation, often affected by external events and structural breaks [3, 23].

Neglecting these dependencies leads to fragmented forecasts and suboptimal inventory policies. This has motivated a shift from isolated univariate models to holistic approaches that exploit structural information, cross-learning, and external data, captured by the notion of *demand sensing*.

1.2. From Classical Forecasting to Graph-Based Demand Sensing

Classical time series models such as exponential smoothing, Holt–Winters, and ARIMA decompose demand into trend, seasonality, and noise components [10]. They remain widely used for their interpretability and speed, particularly on smooth, stationary demand [26]. However, they are ill-suited to modern retail environments, which feature thousands of interdependent series and volatile demand patterns. Three key limitations have been highlighted:

- *Data sparsity and intermittency:* Classical models perform poorly on series with many zeros or erratic spikes, common at the SKU level [22].
- *Lack of cross-learning:* They train separate models per series, ignoring informative correlations between related products or locations [18].
- *Slow adaptation:* They adapt only gradually to structural breaks, promotions, or regime shifts [19].

As a result, forecasts at granular levels often have large errors, while aggregation can hide fine-grained patterns relevant for operations planning [7]. This has spurred interest in *demand sensing*: approaches that integrate external signals and jointly model many related series to produce responsive short-term forecasts.

1.2.1. Demand Sensing and Machine Learning

Demand sensing extends classical forecasting by leveraging diverse external covariates such as prices, promotions, weather, and macroeconomic indicators [21, 1]. Incorporating these drivers has been shown to substantially reduce forecast error, especially during volatile periods [6].

Recent advances in machine learning (ML) enabled demand sensing at scale. Two paradigm shifts are central:

- *Global models:* Instead of fitting one model per series, a single model is trained on all series, allowing data-sparse series to benefit from patterns learned from others [3, 18].
- *Feature-rich inputs:* ML models exploit large sets of engineered features (lags, rolling statistics, calendar effects, external drivers) to capture temporal and contextual signals [16].

Tree-based models (e.g., XGBoost, LightGBM) and neural networks (RNNs, LSTMs, temporal CNNs) have achieved state-of-the-art performance in large-scale forecasting competitions [24, 20, 16]. Yet, these methods typically ignore the hierarchical structure of retail data and produce incoherent forecasts—totals that do not equal the sum of their components—limiting their operational usability.

1.2.2. Hierarchical Forecasting and its Limitations

Many supply chains require forecasts at multiple aggregation levels (e.g., SKU–store–state–national). These levels are linked by linear aggregation constraints. Producing independent forecasts at each level often leads to *incoherency*: the forecasts do not add up correctly [11]. This undermines planning alignment between strategic and operational levels.

Traditional approaches addressed this using reconciliation: base forecasts are generated at all levels and then projected onto the coherent subspace so that they add up correctly. Bottom-up, top-down, and middle-out heuristics [8, 7, 2] and optimal reconciliation methods such as MinT [28] can restore coherence and often improve accuracy. However, they treat structure only as a post-processing constraint rather than a modeling feature, and require estimating large covariance matrices, which is difficult at scale [25]. This motivates end-to-end methods that can *learn* the hierarchy directly during forecasting.

1.2.3. Graphs as a Unifying Representation

Graph-based models offer a natural way to unify cross-series learning and hierarchical structure. A graph can represent time series as nodes and their relationships (correlations, shared attributes, or hierarchy links) as edges, enabling message passing between related series. Recent work on Graph Neural Networks (GNNs) for spatiotemporal forecasting—such as DCRNN [15], STGCN [31], Graph WaveNet [30], and MTGNN [29]—demonstrates how combining graph convolutions with temporal models can scale to thousands of series.

Applying GNNs to hierarchical time series is a recent but promising direction. Hierarchical structures can be represented as tree-structured graphs, allowing information sharing between parent–child and sibling nodes. Rangapuram et al. [25] showed how hierarchy constraints can be embedded directly into deep models, producing probabilistic coherent forecasts by construction. By contrast, since this thesis only focusses on point predictions, our approach encourages coherence during training via a regularization term, while applying reconciliation post-hoc to enforce exact coherence.

1.3. Research Gap and Thesis Objectives

Building on the literature reviewed above, this section outlines the remaining gaps and the objectives of this thesis.

Despite substantial progress in forecasting, current methods remain fragmented. Classical models offer interpretability but ignore cross-series information and external drivers; machine learning models exploit these signals but neglect hierarchical structure, often producing incoherent forecasts; and reconciliation methods restore coherence only as a post-hoc adjustment, without leveraging hierarchical relationships during model training.

Graph neural networks (GNNs) offer a way to unify these perspectives by incorporating hierarchical structure directly into the learning process while jointly modeling cross-series relationships. Yet, their use in large-scale retail demand forecasting remains largely unexplored—particularly regarding how to integrate external drivers while still ensuring coherence.

This thesis addresses these gaps through two guiding research questions:

1. How can hierarchical demand forecasting be formulated as a graph learning problem while ensuring coherent forecasts across all levels?
2. To what extent can graph neural networks improve forecast accuracy compared to classical and machine learning methods, especially when incorporating external drivers?

To answer these questions, the thesis proceeds as follows. **Chapter 2** develops the mathematical foundations of forecasting and hierarchical reconciliation, formalizing coherency constraints and optimal combination methods. **Chapter 3** introduces the proposed graph-based forecasting framework, defining GNN architectures tailored to hierarchical time series. **Chapter 4** applies this framework to the M5 competition dataset, benchmarking its performance against classical and machine learning approaches, both with and without external drivers. Finally, **Chapter 5** synthesizes the findings, discussing their theoretical and practical implications and outlining directions for future research.

2

Forecasting framework

Building on the literature review in Chapter 1, this chapter establishes the theoretical foundations of hierarchical forecasting. The previous chapter highlighted key gaps in how existing methods handle structural dependencies and coherence. To address these gaps, we treat forecasting as a supervised model-based problem, where the objective is to learn a function g_i that maps past observations and external covariates to future demand. We then introduce hierarchical time series structures and the concept of forecast coherence. Together, these form the mathematical backbone for the graph-based approach developed in Chapter 3.

2.1. Time Series Foundations

Demand uncertainty in supply chains necessitates a probabilistic framework, where random quantities are defined with respect to an underlying probability space $(\Omega, \mathcal{F}, P)$. Here Ω represents the set of all possible demand outcomes, $\mathcal{F}$ is a σ-algebra on Ω, and P is a probability measure assigning likelihood to events.

Supply chain events manifest over time, with each period's outcomes influenced by preceding events and external factors. This temporal evolution leads to a time series definition.

Definition 2.1.1 (Time Series)**.** A time series $\{X_t\}_{t\in\mathcal{T}}$ is a sequence of random variables defined on $(\Omega, \mathcal{F}, P)$, where:

$$X_t : (\Omega, \mathcal{F}) \to (\mathbb{R}, \mathcal{B}(\mathbb{R})), \quad t \in \mathcal{T} \tag{2.1}$$

with $\mathcal{B}(\mathbb{R})$ denoting the Borel σ-algebra on $\mathbb{R}$. In most applications, we take $\mathcal{T} = \mathbb{Z}$ or $\mathcal{T} = \mathbb{N}$, representing discrete time periods such as days or weeks.

For forecasting multiple products, we extend this notation where $X_{i,t}$ represents the time series for product $i \in \mathcal{I}$, with $\mathcal{I}$ denoting the set of all products.

Example 2.1.2 (Multi-Product Time Series Dependencies). Consider a retailer tracking demand for three related products: printers ($X_{1,t}$), ink cartridges ($X_{2,t}$), and paper ($X_{3,t}$). Each product's demand follows the time series structure defined above, but their realizations exhibit interdependencies.

During a promotional period for printers at time t^*, we observe:

$$\begin{aligned} X_{1,t^*} &= 150 \text{ units (promotional surge)} \\ X_{2,t^*+1} &= 85 \text{ units (typical: 45 units)} \\ X_{3,t^*+2} &= 230 \text{ units (typical: 180 units)} \end{aligned}$$

The printer promotion at time t^* generates increased demand for complementary products in subsequent periods. Conversely, consider when competitor activity intensifies, where printer sales decrease by 30% while ink cartridge sales increase by 15%, reflecting customer loyalty to existing printer investments despite reduced new printer purchases.

Such interdependencies cannot be captured by treating each $X_{i,t}$ as an independent time series, necessitating frameworks that account for cross-product relationships in both positive (promotional) and negative (competitive) scenarios.

In addition, demand for different products is shaped not only by their past values but also by external factors. Promotions, seasonal effects, or macroeconomic shifts can affect multiple products in distinct ways, amplifying some while dampening others. These external covariates $Z_{i,t}$ must therefore be incorporated alongside cross-product relationships.

2.2. Multi-Product Demand Modeling

Formally, we can express the intuitions listed above as follows.

Definition 2.2.1 (Multi-Product Demand Process)**.** The demand process for a collection of products is defined as:

$$X_{i,t} = g_i(\{(X_{j,t-l}, Z_{j,t-l}) : j \in \mathcal{I}, l = 1, \ldots, L\}; \theta_i) + \varepsilon_{i,t} \quad \forall i \in \mathcal{I} \tag{2.2}$$

where $g_i(\cdot; \theta_i)$ represents the forecasting function for product i with learnable parameters θ_i, $\varepsilon_{i,t}$ represents unpredictable variation in demand not explained by past observations or covariates with

$$\mathbb{E}\left[\varepsilon_{i,t} \mid \{(X_{j,t-l}, Z_{j,t-l}) : j \in \mathcal{I}, l = 1, \ldots, L\}\right] = 0, \tag{2.3}$$

$Z_{j,t}$ captures external covariates for product j, L is the temporal lag order, and $\mathcal{I}$ is the set of all products.

We obtain forecasts by applying the learned forecasting function $g_i(\cdot; \theta_i)$ to the information available up to time t, denoted

$$\mathcal{F}_t = \{(X_{j,s}, Z_{j,s}) : j \in \mathcal{I},\ s \leq t\}.$$

The h-step-ahead forecast for product i at time t can be expressed generically as

$$\hat{X}_{i,t+h|t} = g_i(\{(X_{j,t-l}, Z_{j,t-l}) : j \in \mathcal{I},\ l = 0, \ldots, L\};\ \theta_i)\,.$$

Here, g_i represents a generic forecasting mapping rather than a horizon-specific model. Different formulations exist for generating multi-step forecasts, whether by reusing the same model recursively, training horizon-specific estimators, or predicting all horizons jointly. These distinctions are discussed later in the section on *Multi-Horizon Forecasting*.

For product i and forecast origin t, we denote the H-step sequence of future realizations and their corresponding forecasts as:

$$\boldsymbol{X}_{i,t+1:t+H} := (X_{i,t+1}, \ldots, X_{i,t+H})^\top, \qquad \hat{\boldsymbol{X}}_{i,t+1:t+H|t} := (\hat{X}_{i,t+1|t}, \ldots, \hat{X}_{i,t+H|t})^\top.$$

Collecting these across all horizons yields the H-step forecast vector $\hat{\boldsymbol{X}}_{i,t+1:t+H|t}$ defined above.

The multi-product demand process in Equation (2.2) involves two central design choices that shape both forecasting accuracy and computational cost.

- *Information Scope:* For each product i, the function $g_i(\cdot; \theta_i)$ may use inputs from all products ($j \in \mathcal{I}$) or from a selected subset ($j \in \mathcal{S}_i \subseteq \mathcal{I}$). A wide scope captures more cross-product dependencies but increases input dimensionality and training cost, while a narrower scope may improve efficiency at the expense of missing relevant signals.
- *Parameter Sharing:* Each product can have its own parameter set θ_i (local models) or share a common parameter set θ across all products (a global model). Local models can capture product-specific dynamics but require substantial data per series, whereas global models enable cross-learning between products but risk underfitting product-level idiosyncrasies.

These decisions lead to two primary modeling strategies.

2.2.1. 1ML-ForEach Strategy

The 1ML-ForEach approach uses product-specific functions with selective cross-product information:

$$X_{i,t} = g_i\left(\{(X_{j,t-l}, Z_{j,t-l}) : j \in \mathcal{S}_i, l = 1, \ldots, L\}; \theta_i\right) + \varepsilon_{i,t} \tag{2.4}$$

where $\mathcal{S}_i \subseteq \mathcal{I}$ represents the subset of products relevant for forecasting product i, and each g_i has its own parameter set θ_i. This approach ranges from univariate modeling ($\mathcal{S}_i = \{i\}$) to full cross-product modeling ($\mathcal{S}_i = \mathcal{I}$), enabling parallel training.

2.2.2. 1ML-ForAll Strategy

The 1ML-ForAll approach uses a shared function across all products with product-specific embeddings:

$$X_{i,t} = g\left(\mathcal{E}_i, \{(X_{j,t-l}, Z_{j,t-l}) : j \in \mathcal{I}, l = 1, \ldots, L\}; \theta_i\right) + \varepsilon_{i,t} \tag{2.5}$$

where $\mathcal{E}_i \in \mathbb{R}^d$ encodes product i's characteristics and $g(\cdot)$ shares parameters θ across all products. This enables cross-product learning where patterns from one product inform predictions for others, but requires sufficient data across all products.

2.2.3. Multi-Horizon Forecasting

As introduced earlier, different formulations can be used to generate forecasts over multiple horizons. In supply chains, decisions are horizon-dependent: short-term store replenishment, medium-term distribution planning, and long-term supplier negotiations all require forecasts with different lead times. Formally, horizons are indexed by $h \in \{1, \ldots, H\}$.

Two main strategies operationalize horizon-specific forecasts:

Definition 2.2.2 (Iterative (Recursive) Forecast)**.** Fix a lag order L and a product-specific information set $\mathcal{S}_i \subseteq \mathcal{I}$. For horizons $h = 1, \ldots, H$, the iterative forecast reuses the same estimator g_i recursively:

$$\hat{X}_{i,t+h|t} = g_i\Big(\Big\{\big(\tilde{X}_{j,t+h-\ell|t},\ \tilde{Z}_{j,t+h-\ell|t}\big) : j \in \mathcal{S}_i,\ \ell = 1, \ldots, L\Big\}\Big),$$

where $\tilde{X}_{j,\tau|t} := X_{j,\tau}$ if $\tau \leq t$ (observed) and $\tilde{X}_{j,\tau|t} := \hat{X}_{j,\tau|t}$ if $\tau > t$ (previously predicted).
The covariates $\tilde{Z}_{j,\tau|t}$ denote the exogenous variables available at forecast origin t. When future covariates are unknown, only lagged or time-invariant covariates are used, i.e. $\tilde{Z}_{j,\tau|t} := Z_{j,\tau}$ for $\tau \leq t$. If certain covariates are known in advance (e.g., calendar effects or planned promotions), their future values can be incorporated accordingly.

Remark. In our experiments, we follow the latter approach: future covariates are not assumed to be known, so recursive forecasts rely only on lagged or deterministic covariates such as calendar indicators and product attributes. This ensures that all inputs used at prediction time reflect information available at or before t.

Definition 2.2.3 (Direct h-Step Forecast)**.** The direct approach trains a separate horizon-specific mapping $g_{i,h}$ for each $h = 1, \ldots, H$:

$$\hat{X}_{i,t+h|t} = g_{i,h}\big(\big\{\big(X_{j,t-\ell},\ Z_{j,t-\ell}\big) : j \in \mathcal{S}_i,\ \ell = 1, \ldots, L\big\}\big).$$

Each $g_{i,h}$ is optimized to predict h steps ahead using only information available at time t, avoiding recursive error accumulation but requiring multiple horizon-specific estimators.

The choice between iterative and direct approaches interacts with model architecture. Iterative forecasts with shared g (1ML-ForAll) naturally propagate cross-product information through time, while direct forecasts require explicit horizon-specific modeling of cross-product effects.

2.3. Estimators for Multi-Product Systems

Estimating the multi-product demand process in equation (2.2) requires learning forecasting functions $\hat{g}_i(\cdot; \hat{\theta}_i)$ that map historical observations to future demand.

Definition 2.3.1 (Forecast Operator)**.** Let the multi-product demand process be

$$X_{i,t} = g_i\big(\{(X_{j,t-l}, Z_{j,t-l}) : j \in \mathcal{I}, l \le L\}\big) + \varepsilon_{i,t}, \qquad i \in \mathcal{I}.$$

For product i and forecast origin t, define the available information as

$$\mathcal{F}_t = \{(X_{j,s}, Z_{j,s}) : j \in \mathcal{I}, s < t\}.$$

Given a learned forecasting function $\hat{g}_i(\cdot;\hat{\theta}_i)$ (for 1ML-ForEach) or $\hat{g}_\theta(\cdot)$ (for 1ML-ForAll), the H-step *forecast operator* produces the forecast vector

$$\mathcal{F}_{\hat{g}_i}(\mathcal{F}_t) := \hat{\boldsymbol{X}}_{i,t+1:t+H|t} = \big(\hat{X}_{i,t+1|t}, \ldots, \hat{X}_{i,t+H|t}\big)^\top,$$

where each step prediction is recursively or directly computed from $\mathcal{F}_t$ using the learned model, without the noise term $\varepsilon_{i,t}$.

This framing lets us interpret any forecasting method as a specific implementation of the forecast operator for the multi-product demand process. We now examine these estimators in detail, describing for each how it instantiates the multi-product demand process through five aspects:

1. its formulation within the demand process.
2. its design choice of information scope and parameter sharing.
3. its concrete model form.
4. how its forecasts are produced via the forecast operator.
5. its overall position in terms of strengths and limitations.

2.3.1. Separate Estimators (1ML-ForEach)

ARIMA / ETS (Univariate Models). These classical approaches model the demand process for each product independently, i.e. with $\mathcal{S}_i = \{i\}$, using only the history of series i:

$$X_{i,t} = g_i\big(\{(X_{i,t-l}, Z_{i,t-l}) : l = 1, \ldots, L\}\big) + \varepsilon_{i,t}.$$

Each product has its own parameter set θ_i, so no information is shared across series.

A typical ARIMA specification is

$$g_i(\cdot) = \sum_{l=1}^{p} \phi_{i,l} X_{i,t-l} + \sum_{l=1}^{q} \eta_{i,l} \hat{\varepsilon}_{i,t-l} + \gamma_i^\top Z_{i,t-l}, \quad \theta_i = \big(\{\phi_{i,l}\}, \{\eta_{i,l}\}, \gamma_i\big),$$

where the autoregressive terms (ϕ) use past observations $X_{i,t-l}$, and the moving-average terms (η) depend on previously estimated residuals $\hat{\varepsilon}_{i,t-l}$ rather than unobservable noise terms.

These residuals are obtained recursively during model estimation as

$$\hat{\varepsilon}_{i,t} = X_{i,t} - \hat{g}_i\big(\{(X_{i,t-l}, Z_{i,t-l}) : l = 1, \ldots, L\}; \hat{\theta}_i\big),$$

so that only past residual estimates $\hat{\varepsilon}_{i,t-l}$ are available at forecast origin t.

Given fitted parameters $\hat{\theta}_i$, the forecast operator $\mathcal{F}_{\hat{g}_i}$ produces iterative forecasts $\hat{X}_{i,t+h|t}$ based solely on lagged observations $(X_{i,t-1}, \ldots, X_{i,t-L})$ and past residual estimates. These models are highly interpretable and scale easily to many series, but they ignore cross-series relationships entirely.

TBATS (Multiple-Seasonality Model). The TBATS model [5] extends the exponential smoothing (ETS) framework to capture complex seasonal patterns and nonlinear trends that often arise in retail demand data.

It instantiates the demand process for each product independently:

$$X_{i,t} = g_i\big(\{(X_{i,t-l}, Z_{i,t-l}) : l = 1, \ldots, L\}\big) + \varepsilon_{i,t},$$

with $g_i(\cdot)$ specified as a nonlinear state-space function that combines Box–Cox transformation, ARMA error components, and multiple trigonometric seasonal terms.

Formally, the observation equation can be written as

$$X_{i,t}^{(\lambda_i)} = \ell_{i,t-1} + \phi_i b_{i,t-1} + \sum_{k=1}^{K_i} s_{i,t-1}^{(k)} + d_{i,t}, \qquad d_{i,t} \sim \mathsf{ARMA}(p_i, q_i),$$

where:

- $X_{i,t}^{(\lambda_i)}$ is the Box–Cox transformed observation with parameter λ_i,
- $\ell_{i,t}$ and $b_{i,t}$ denote the local level and trend,
- $s_{i,t}^{(k)}$ represents the k-th seasonal component, modelled via Fourier terms with period m_k,
- $d_{i,t}$ is the short-term ARMA disturbance term capturing residual autocorrelation. During forecasting, only past residual estimates $\hat{d}_{i,t-l}$ are available within the information set $\mathcal{F}_t$, and future innovations are assumed to have conditional mean zero, ensuring

$$\mathbb{E}[d_{i,t+h} \mid \mathcal{F}_t] = 0, \quad \forall h > 0.$$

Each component evolves over time according to a set of smoothing parameters governing level, trend, and seasonality updates. The model therefore adapts to nonstationary seasonalities, such as weekly and yearly cycles, and can incorporate exogenous regressors $Z_{i,t}$ through additive regression terms:

$$g_i(\cdot) = \mathsf{TBATS}\big(X_{i,t-1:t-L},\, Z_{i,t-1:t-L};\, \theta_i\big), \quad \theta_i = \{\lambda_i, \phi_i, p_i, q_i, K_i, m_k\}.$$

Given fitted parameters $\hat{\theta}_i$, the forecast operator $\mathcal{F}_{\hat{g}_i}$ generates multi-step forecasts by propagating the state-space equations forward in time, producing $\hat{X}_{i,t+h|t}$ for horizons $h = 1{:}H$.

TBATS retains interpretability similar to ETS while handling long and multiple seasonal cycles, making it particularly suitable for products with calendar effects and holiday-driven demand fluctuations. However, like other univariate estimators, it does not share information across products.

Product-Specific Neural Networks. These models implement the demand process using a local information scope $\mathcal{S}_i$ around product i:

$$X_{i,t} = f_{\theta_i}\big(\{(X_{j,t-l}, Z_{j,t-l}) : j \in \mathcal{S}_i,\, l \le L\}\big) + \varepsilon_{i,t}.$$

Each product i has its own neural network f_{θ_i} with parameter set $\theta_i = \{W_i^{(1)}, \ldots, W_i^{(D)}, b_i^{(1)}, \ldots, b_i^{(D)}\}$, where $W_i^{(d)}$ and $b_i^{(d)}$ denote the weights and biases of the d-th layer and D is the total number of layers.

The network f_{θ_i} maps the concatenated lagged inputs $\boldsymbol{x}_{i,t} = \{(X_{j,t-l}, Z_{j,t-l}) : j \in \mathcal{S}_i, l \le L\}$ to a forecast as

$$\begin{aligned} \boldsymbol{h}_i^{(0)} &= \boldsymbol{x}_{i,t}, \\ \boldsymbol{h}_i^{(d)} &= \sigma\big(W_i^{(d)} \boldsymbol{h}_i^{(d-1)} + b_i^{(d)}\big) \quad \forall d = 1, \ldots, D, \\ f_{\theta_i}(\boldsymbol{x}_{i,t}) &= W_i^{(D+1)} \boldsymbol{h}_i^{(D)} + b_i^{(D+1)}, \end{aligned}$$

where $\sigma(\cdot)$ is a nonlinear activation function such as ReLU or tanh.

Given fitted parameters $\hat{\theta}_i$, the forecast operator $\mathcal{F}_{f_{\hat{\theta}_i}}(\cdot)$ produces H-step forecasts

$$\hat{\boldsymbol{X}}_{i,t+1:t+H|t} = \mathcal{F}_{f_{\hat{\theta}_i}}\big(\boldsymbol{x}_{i,t}\big),$$

where each horizon h is predicted from the most recent L-step input window. This design enables rich nonlinear interactions and local specialization, but comes with high parameter counts and substantial overfitting risk when data per product is limited.

2.3.2. Shared Estimators (1ML-ForAll)

Vector Autoregression (VAR). VAR extends the demand process to a fully multivariate setting, modeling all products jointly with shared parameters:

$$\boldsymbol{X}_t = g_\theta\big(\boldsymbol{X}_{t-1:t-L}, \boldsymbol{Z}_t\big) + \boldsymbol{\varepsilon}_t.$$

A single global linear model captures cross-series dependencies through lagged interactions across all products. Its functional form is

$$\boldsymbol{X}_t = \boldsymbol{A}_1\boldsymbol{X}_{t-1} + \cdots + \boldsymbol{A}_L\boldsymbol{X}_{t-L} + \boldsymbol{B}\boldsymbol{Z}_t + \boldsymbol{\varepsilon}_t, \quad \theta = \{\boldsymbol{A}_1, \ldots, \boldsymbol{A}_L, \boldsymbol{B}\},$$

where each $\boldsymbol{A}_l$ encodes cross-lag effects between products and $\boldsymbol{B}$ maps external drivers.

Given estimated parameters $\hat{\theta}$, the forecast operator $\mathcal{F}_{\hat{g}}$ generates multi-step forecasts recursively as

$$\hat{\boldsymbol{X}}_{t+1:t+H|t} = \mathcal{F}_{\hat{g}}\big(\boldsymbol{X}_{t:t-L+1}, \boldsymbol{Z}_{t:t-L+1}\big),$$

where each forecasted $\hat{\boldsymbol{X}}_{t+h|t}$ is fed back as input for the next step.

VAR effectively captures linear cross-series dynamics, but its parameter count grows quadratically with the number of products, which limits scalability in large retail settings.

Global Machine Learning Models (LightGBM, LSTM, Transformer). Global ML models extend the demand process by training a single forecasting function on all products jointly, using product embeddings $\mathcal{E}_i$ to distinguish series:

$$X_{i,t} = g_\theta\big(\mathcal{E}_i, \{(X_{j,t-l}, Z_{j,t-l})\}_{j\in\mathcal{I}, l\le L}\big) + \varepsilon_{i,t}.$$

This design enables full parameter sharing: information from every product contributes to forecasting each individual series.
The forecasting function is implemented as

$$g_\theta(\mathcal{E}_i, \cdot) = f_\theta\big(\mathcal{E}_i, \{(X_{j,t-l}, Z_{j,t-l})\}_{j\in\mathcal{I}, l\le L}\big),$$

where f_θ can take different forms depending on the model class.

For example, in LightGBM (a gradient-boosted decision tree ensemble), f_θ is a collection of regression trees whose structure is learned from data to minimize prediction error, and predictions are obtained by summing the outputs of all trees. In LSTM or Transformer models, f_θ is a deep neural network that applies multiple layers of nonlinear transformations (with parameters such as weight matrices and biases) to the input sequence, using mechanisms such as recurrence or self-attention to model long-range temporal dependencies.
The embedding vector $\mathcal{E}_i$ provides the model with a fixed-length representation of the product identity. It can be constructed in several ways:

- *Learned embeddings:* A lookup table of trainable vectors $\mathcal{E}_i \in \mathbb{R}^d$ is optimized jointly with the rest of the model parameters θ.
- *Attribute-based embeddings:* Metadata such as product category, brand, price range, or store location can be concatenated and passed through a small neural network to obtain $\mathcal{E}_i$.

Given trained parameters $\hat{\theta}$, the forecast operator produces multi-step forecasts as

$$\hat{X}_{i,t+h|t} = \mathcal{F}_{\hat{g}}\big(\mathcal{E}_i, \boldsymbol{X}_{t:t-L+1}, \boldsymbol{Z}_{t:t-L+1}\big),$$

where predicted values are recursively fed back as inputs to forecast future horizons.
Such global models achieve strong accuracy by exploiting cross-series learning and shared structure, but they ignore hierarchical relationships and typically produce incoherent forecasts.

Linear State-Space Networks (LSTFLinear / LSTFNLinear). State-space networks refer to deterministic neural architectures inspired by linear state-space dynamics (**?**). The abbreviation LSTF refers to *Linear State-space Time-series Forecaster*, a model that represents temporal dynamics through latent linear state transitions. The variant LSTFN extends this structure with learned *N*ormalization layers, introducing additional flexibility in how latent states interact with inputs.

Both architectures instantiate the demand process using latent states that evolve over time:

$$\boldsymbol{X}_t = g_\theta(\boldsymbol{X}_{t-1:t-L}, \boldsymbol{Z}_{t-1:t-L}) + \boldsymbol{\varepsilon}_t.$$

They adopt a global design: a single model is trained on all products jointly, capturing cross-series dependencies through shared latent dynamics. The forecasting function is realized as

$$f_\theta : (\boldsymbol{X}_{t-1:t-L}, \boldsymbol{Z}_{t-1:t-L}) \mapsto \boldsymbol{X}_t,$$

where f_θ follows a (possibly nonlinear) state-space structure parameterized by $\theta = \{\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}\}$.

In the **LSTFLinear** case, the transition and emission mappings are strictly linear:

$$\boldsymbol{h}_t = \boldsymbol{A}\boldsymbol{h}_{t-1} + \boldsymbol{B}\boldsymbol{Z}_t, \qquad \boldsymbol{X}_t = \boldsymbol{C}\boldsymbol{h}_t + \boldsymbol{\nu}_t.$$

In contrast, the **LSTFNLinear** variant introduces nonlinearities in the state transition or output mapping, e.g. through elementwise activations or neural gating mechanisms. All products share the same parameters θ, allowing the model to couple their dynamics through a common low-dimensional representation $\boldsymbol{h}_t$.

Given fitted parameters $\hat{\theta}$, the forecast operator produces multi-step forecasts as

$$\hat{\boldsymbol{X}}_{t+1:t+H|t} = \mathcal{F}_{\hat{g}}\big(\boldsymbol{h}_t, \boldsymbol{Z}_{t+1:t+H}\big),$$

by rolling out the latent state forward via

$$\boldsymbol{h}_{t+h} = \boldsymbol{A}^h \boldsymbol{h}_t + \sum_{r=1}^{h} \boldsymbol{A}^{h-r}\boldsymbol{B}\boldsymbol{Z}_{t+r}, \qquad \hat{\boldsymbol{X}}_{t+h|t} = \boldsymbol{C}\boldsymbol{h}_{t+h}.$$

This taxonomy shows how seemingly different estimators are all realizations of the same multi-product demand process, differing only in their choice of information scope and parameter sharing. Later chapters evaluate how these design choices affect forecast accuracy.

2.3.3. Evaluating Forecast Quality

Having specified the forecasting function $g_i(\cdot)$ through statistical estimators, the next question is how to evaluate the resulting forecasts $\hat{X}_{i,t+h|t}$. In the multi-product demand process, forecast quality must be assessed both at the individual product level and at the portfolio level.

Definition 2.3.2 (Expected Risk)**.** Let $\ell(\cdot,\cdot)$ be a pointwise (per-horizon) loss. The horizon-aggregated loss is

$$\mathcal{L}\big(\boldsymbol{X}_{i,t+1:t+H}, \hat{\boldsymbol{X}}_{i,t+1:t+H|t}\big) := \frac{1}{H}\sum_{h=1}^{H} \ell\big(X_{i,t+h}, \hat{X}_{i,t+h|t}\big),$$

and the expected risk is

$$\mathcal{R}_i\big(\hat{\boldsymbol{X}}_{i,t+1:t+H|t}\big) := \mathbb{E}\Big[\mathcal{L}\big(\boldsymbol{X}_{i,t+1:t+H}, \hat{\boldsymbol{X}}_{i,t+1:t+H|t}\big)\Big].$$

Different choices of $\mathcal{L}$ correspond to different operational assumptions. Importantly, the suitability of a loss function depends on how forecasts interact across products $i \in \mathcal{I}$.

Mean Squared Error (MSE):

$$\ell_{\text{MSE}}\big(X_{i,t+h}, \hat{X}_{i,t+h|t}\big) = (X_{i,t+h} - \hat{X}_{i,t+h|t})^2.$$

MSE implements a quadratic penalty, making it natural when over- and under-forecasting have symmetric but compounding effects (e.g., capacity planning for complementary products). However, in multi-product portfolios with heterogeneous demand, MSE tends to overweight high-volume products.

Mean Absolute Error (MAE):

$$\ell_{\text{MAE}}\big(X_{i,t+h}, \hat{X}_{i,t+h|t}\big) = \big|X_{i,t+h} - \hat{X}_{i,t+h|t}\big|.$$

MAE corresponds to a linear penalty and is robust to outliers. It treats all deviations equally, making it attractive when comparing strategies such as 1ML-ForEach versus 1ML-ForAll, since it avoids letting high-volume products dominate the portfolio loss.

Mean Absolute Percentage Error (MAPE):

$$\ell_{\text{MAPE}}\big(X_{i,t+h}, \hat{X}_{i,t+h|t}\big) = \left|\frac{X_{i,t+h} - \hat{X}_{i,t+h|t}}{X_{i,t+h}}\right| \times 100\%.$$

MAPE normalizes errors by product scale, enabling fairer comparison across products with very different demand levels. This property is crucial when assessing global models (1ML-ForAll) where the same g is shared across high- and low-volume products. However, MAPE becomes unstable when $x \approx 0$, making it less suitable for intermittent demand.

Weighted Root Mean Squared Scaled Error (WRMSSE): WRMSSE extends the MSE by applying two normalizations: (i) scaling each series by its in-sample mean squared first difference to remove scale effects, and (ii) weighting each series by its relative sales volume to reflect operational importance.

$$\ell_{\text{WRMSSE}}\big(X_{i,t+h}, \hat{X}_{i,t+h|t}\big) = w_i \, \frac{(X_{i,t+h} - \hat{X}_{i,t+h|t})^2}{s_i},$$

where $w_i \geq 0$ are volume weights with $\sum_{i \in \mathcal{I}} w_i = 1$, and the scale term (computed in-sample) is

$$s_i = \frac{1}{T_i - 1} \sum_{\tau=2}^{T_i} \big(X_{i,\tau} - X_{i,\tau-1}\big)^2.$$

Aggregating over horizons and series gives

$$\mathcal{L}_{\text{WRMSSE}} = \sqrt{\frac{1}{H} \sum_{h=1}^{H} \sum_{i \in \mathcal{I}} \ell_{\text{WRMSSE}}\big(X_{i,t+h}, \hat{X}_{i,t+h|t}\big)}.$$

This metric was used in the M5 competition to evaluate forecasts across multiple time series at multiple aggregation levels.

Loss Functions in Multi-Product Contexts

The choice of loss function is not neutral: it interacts directly with the structure of g_i.

- In **1ML-ForEach**, separate models are estimated per product. Losses are naturally computed individually, but aggregating them into a portfolio score requires weighting (equal-weight vs volume-weight).
- In **1ML-ForAll**, a single shared g is trained. Losses must ensure that smaller products are not overshadowed by larger ones, making normalized measures (e.g., MAPE, WRMSSE) more suitable.

2.4. From Individual to Hierarchical Forecasting

The forecasting framework we have established provides a solid foundation for individual time series. However, in modern supply chains, demand processes rarely exist in isolation. Consider a retail organization where:

- Products are grouped into categories and departments
- Stores are clustered by regions and countries
- Demand is aggregated across different time periods

This natural hierarchical structure introduces additional complexities:

$$X_t^{(\text{total})} = \sum_{i \in \mathcal{I}} X_{i,t} = \sum_{r \in \mathcal{R}} \sum_{s \in \mathcal{S}_r} \sum_{p \in \mathcal{P}_s} X_{(r,s,p),t}, \tag{2.6}$$

where $\mathcal{R}$ represents regions, $\mathcal{S}_r$ stores in region r, and $\mathcal{P}$ products.
The challenge extends beyond simply applying our forecasting framework at each level independently. We must ensure:

- Coherence: Forecasts at different levels should aggregate consistently
- Information Flow: Patterns detected at one level inform forecasts at other levels
- Efficiency: Computational methods scale with hierarchy size

These considerations lead us to develop a hierarchical forecasting framework, which we explore in the next section. In the next section, we will replace the more theoretical notation of a forecast $\hat{X}_{t+h|t}$ with a more practical notation $\hat{y}_{t+h}$.

2.5. Hierarchical Forecasting

In many real-world applications, data is naturally organized into hierarchical structures where observations can be grouped and aggregated across different levels. The mathematical treatment of such structures requires careful consideration of both the hierarchical relationships and the constraints they impose on the forecasts.

2.5.1. Hierarchical Structure and Coherency

The hierarchical structure naturally takes the form of a tree, where each element represents a node that can be aggregated with other nodes at a lower level. Starting from the top-most level, which contains a single node representing the most aggregated data, the structure branches down through intermediate levels, with each node potentially splitting into multiple nodes at the next lower level.

Definition 2.5.1 (Hierarchical Structure)**.** Consider a hierarchical structure with L levels. For any level $l \in \{1, \dots, L\}$, let $m_l \in \mathbb{N}$ denote the number of nodes at level l, where $l = 1$ represents the most aggregated level and $l = L$ the most disaggregated level. At each level l, we define a forecast vector $\hat{\boldsymbol{y}}_t^{(l)} \in \mathbb{R}^{m_l}$, consisting of individual forecasts $\hat{y}_{i,t}^{(l)} := (\hat{\boldsymbol{y}}_t^{(l)})_i \in \mathbb{R}$ for each node $i \in \mathcal{I}^{(l)}$.

The hierarchical relationships between levels are encoded through sets $\mathcal{I}_i^{(l)}$ for each node $i \in \mathcal{I}^{(l)}$, where $\mathcal{I}_i^{(l)}$ contains the indices of all children of node i at level $l+1$. These sets satisfy:

$$\mathcal{I}^{(l+1)} = \biguplus_{i \in \mathcal{I}^{(l)}} \mathcal{I}_i^{(l)} \tag{2.7}$$

where $\mathcal{I}^{(l+1)}$ represents all nodes at level $l+1$.

Example 2.5.2 (Retail Sales Hierarchy)*.* Consider a company with two regions, where the first region contains three stores and the second region contains two stores. This gives us a three-level hierarchy where:

- Level 1 ($l = 1$): Company total ($m_1 = 1$)
- Level 2 ($l = 2$): Regional totals ($m_2 = 2$)

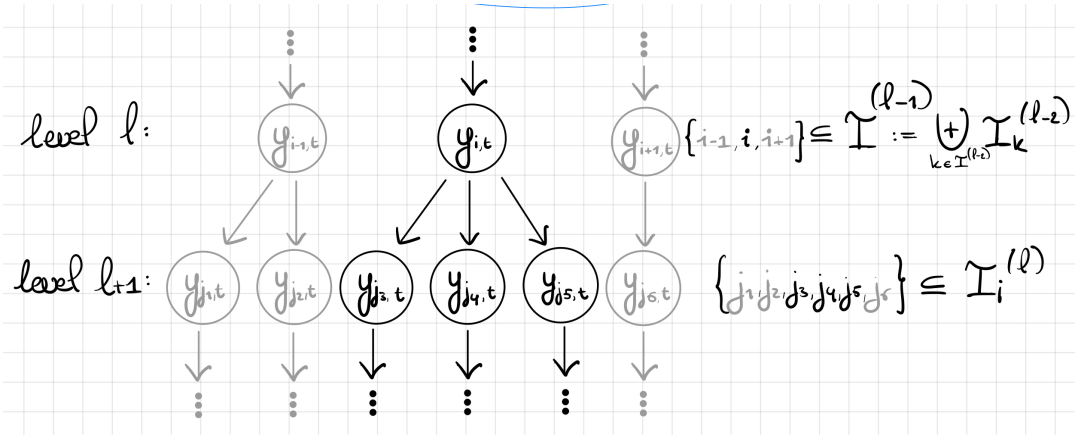


Figure 2.1: Illustration of a hierarchical forecasting structure. At level l, each node $\{i-1, i, i+1\} \subseteq \mathcal{I}^{(l-1)}$ corresponds to their respective forecast $\left\{\hat{y}^{(l)}_{i-1,t}, \hat{y}^{(l)}_{i,t}, \hat{y}^{(l)}_{i+1,t}\right\}$, which together are part of the vector $\hat{\boldsymbol{y}}^{(l)}_t \in \mathbb{R}^{m_l}$. The sets $\mathcal{I}^{(l)}_i$ encode parent-child relationships between levels, where each set contains the indices of children at level $l+1$ for nodes $\{i-1, i, i+1\}$.

- Level 3 ($l = 3$): Store-level sales ($m_3 = 5$)

This example will serve as a running illustration as we develop the formal theory.

Remark. The hierarchical structure defined in Definition 2.5.1 has the following properties:

- For any level $l = 1, \ldots, L-1$, the number of nodes at level $l+1$ equals the sum of the cardinalities of all children sets at level l:
$$m_{l+1} = \sum_{i \in \mathcal{I}^{(l)}} |\mathcal{I}^{(l)}_i|$$
- For the top-most level ($l = 1$), we have $m_1 = 1$ and $\mathcal{I}^{(1)} = \{1\}$
- The sets $\mathcal{I}^{(l)}_i$ are disjoint for any fixed level l, i.e.,
$$\mathcal{I}^{(l)}_i \cap \mathcal{I}^{(l)}_j = \emptyset \quad \text{for } i \neq j$$
- The dimension of each forecast vector increases as we move down the hierarchy:
$$1 = m_1 \leq m_2 \leq \cdots \leq m_L$$

To ensure coherence between the levels of the hierarchical structure, an aggregation constraint must be imposed. This means that the forecast at node i and any level l in the hierarchy must be the sum of the forecasts at the level below it. This aggregation constraint between the two levels naturally follow a parent-child relationships, which we encode in our hierarchical structure through the sets $\mathcal{I}^{(l)}_i$. While this constraint can be expressed for any pair of adjacent levels, true coherency requires this relationship to hold consistently throughout the entire hierarchy. This leads us to formalize the notion of coherency in the following definition.

Definition 2.5.3 (Coherency)**.** For any level $l \in \{1, \ldots, L-1\}$ and node $i \in \mathcal{I}^{(l)}$, we can express local coherency by stating that the forecast at a parent node must equal the sum of forecasts of all its children nodes:

$$\hat{y}^{(l)}_{i,t} = \sum_{j \in \mathcal{I}^{(l)}_i} \hat{y}^{(l+1)}_{j,t} \tag{2.8}$$

We say the entire hierarchy $\{\hat{y}^{(l)}_t\}_{l=1}^{L}$ is globally coherent if this relationship holds simultaneously for all nodes at all levels:

$$\hat{y}^{(l)}_{i,t} = \sum_{j \in \mathcal{I}^{(l)}_i} \hat{y}^{(l+1)}_{j,t}, \quad \forall i \in \mathcal{I}^{(l)}, \quad \forall l \in \{1, \ldots, L-1\} \tag{2.9}$$

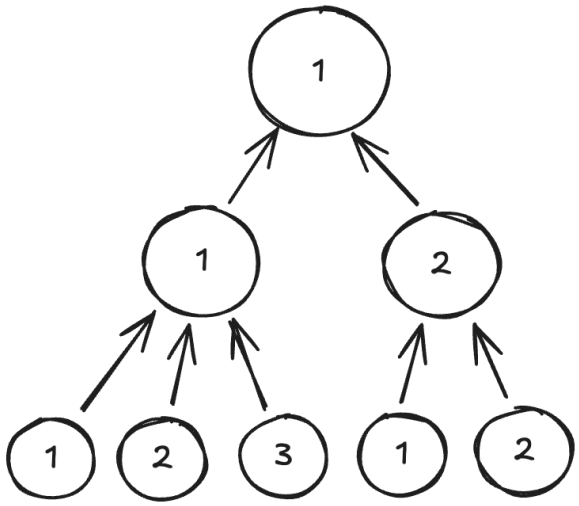


Figure 2.2: Three-level hierarchical structure (L = 3) with the corresponding indices at each level. The arrows indicate the aggregation relationships between levels, where each parent node's forecast must equal the sum of its children nodes' forecasts.

Example 2.5.4 (Continued: Three-Level Hierarchy). Building on our company example, we can now express the hierarchical structure using our formal notation. Recall that we have a company with two regions, where the first region contains three stores and the second region contains two stores. Specifically:

- At the bottom level $(l = 3)$: The five stores are represented by $m_3 = 5$ nodes with forecasts $\{\hat{y}_{i,t}^{(3)}\}_{i\in\mathcal{I}^{(3)}}$, where $\mathcal{I}^{(3)} = \{1, 2, 3, 4, 5\}$
- At the regional level $(l = 2)$: The two regions correspond to $m_2 = 2$ nodes with forecasts $\{\hat{y}_{i,t}^{(2)}\}_{i\in\mathcal{I}^{(2)}}$, where $\mathcal{I}^{(2)} = \{1, 2\}$
- At the company level $(l = 1)$: The total company sales are represented by $m_1 = 1$ node with forecast $\hat{y}_{1,t}^{(1)}$

The parent-child relationships between regions and stores are encoded through the sets:

$$\begin{aligned} \mathcal{I}_1^{(1)} &= \{1, 2\} && \text{(regions belonging to the company)} \\ \mathcal{I}_1^{(2)} &= \{1, 2, 3\} && \text{(stores in first region)} \\ \mathcal{I}_2^{(2)} &= \{4, 5\} && \text{(stores in second region)} \end{aligned} \tag{2.10}$$

When the forecasts are coherent throughout this hierarchy, we have:

$$\begin{aligned} \hat{y}_{1,t}^{(1)} &= \hat{y}_{1,t}^{(2)} + \hat{y}_{2,t}^{(2)} && \text{(company total = sum of regions)} \\ \hat{y}_{1,t}^{(2)} &= \hat{y}_{1,t}^{(3)} + \hat{y}_{2,t}^{(3)} + \hat{y}_{3,t}^{(3)} && \text{(first region = sum of its stores)} \\ \hat{y}_{2,t}^{(2)} &= \hat{y}_{4,t}^{(3)} + \hat{y}_{5,t}^{(3)} && \text{(second region = sum of its stores)} \end{aligned} \tag{2.11}$$

Following these coherency relationships through the hierarchy, we can express the company's total sales in terms of all store-level sales:

$$\hat{y}_{1,t}^{(1)} = \sum_{i\in\mathcal{I}^{(3)}} \hat{y}_{i,t}^{(3)} = \hat{y}_{1,t}^{(3)} + \hat{y}_{2,t}^{(3)} + \hat{y}_{3,t}^{(3)} + \hat{y}_{4,t}^{(3)} + \hat{y}_{5,t}^{(3)} \tag{2.12}$$

The example demonstrates how the formal definitions capture the natural aggregation relationships in a business hierarchy, and how coherency ensures consistent forecasts across all organizational levels. The recursive relationship in equation (2.12) demonstrates an important property: we can express forecasts at any level in terms of forecasts at lower levels by repeatedly applying the coherency constraints.

This observation suggests that coherency has a transitive nature: if forecasts are coherent between consecutive levels, they should also be coherent between any two levels in the hierarchy, not just adjacent ones. This insight leads us to formalize how local coherency between consecutive levels propagates through the hierarchy. By carefully examining the algebraic structure of these relationships, we can establish the following theorem that precisely characterizes this recursive property.

Theorem 2.5.5 (Recursive Coherency)**.** For any level $l \in \{1, \dots, L-2\}$ and node $i \in \mathcal{I}^{(l)}$, if coherency holds locally between adjacent levels l to $l+1$ and $l+1$ to $l+2$, then there exists a direct local coherency relationship between levels l and $l+2$ given by:

$$\hat{y}_{i,t}^{(l)} = \sum_{k \in \biguplus_{j \in \mathcal{I}_i^{(l)}} \mathcal{I}_j^{(l+1)}} \hat{y}_{k,t}^{(l+2)} \tag{2.13}$$

Proof. Fix a level l and node $i \in \mathcal{I}^{(l)}$. By the coherency between levels l and $l+1$:

$$\hat{y}_{i,t}^{(l)} = \sum_{j \in \mathcal{I}_i^{(l)}} \hat{y}_{j,t}^{(l+1)} \tag{2.14}$$

For each $j \in \mathcal{I}_i^{(l)}$, by coherency between levels $l+1$ and $l+2$:

$$\hat{y}_{j,t}^{(l+1)} = \sum_{k \in \mathcal{I}_j^{(l+1)}} \hat{y}_{k,t}^{(l+2)} \tag{2.15}$$

Substituting the second equation into the first:

$$\hat{y}_{i,t}^{(l)} = \sum_{j \in \mathcal{I}_i^{(l)}} \left(\sum_{k \in \mathcal{I}_j^{(l+1)}} \hat{y}_{k,t}^{(l+2)} \right) \tag{2.16}$$

By the properties of our hierarchical structure and the disjoint nature of the children sets $\mathcal{I}_j^{(l+1)}$, we can rewrite this double summation as a single sum over the union of all children sets

$$\hat{y}_{i,t}^{(l)} = \sum_{k \in \biguplus_{j \in \mathcal{I}_i^{(l)}} \mathcal{I}_j^{(l+1)}} \hat{y}_{k,t}^{(l+2)} \tag{2.17}$$

which concludes the proof. □

The nested summations and set operations in the above theorem, while correct, can become unwieldy when working with complex hierarchical structures or when developing further theoretical results. Particularly, the recursive application of coherency through the expression in equation (2.13) suggests that an alternative formulation might be more appropriate. By encoding these relationships in matrix form, we can leverage linear algebra to express these same concepts more concisely and derive results more efficiently. This motivates the following alternative definition of our hierarchical structure using summing matrices $\boldsymbol{S}_l$.

2.5.2. Matrix Representation of Hierarchical Forecasting

In hierarchical forecasting, the aggregation across levels involves summing forecasts at more granular levels to obtain forecasts at more aggregated levels. While the summing notation explicitly expresses how each lower-level forecast contributes to the higher level, matrix notation allows for a more compact and efficient representation of this process, especially when dealing with large hierarchies.

Definition 2.5.6 (Summing Matrix)**.** For any level $l \in \{1, \dots, L-1\}$, let $\boldsymbol{S}_l \in \{0,1\}^{m_l \times m_{l+1}}$ be the summing matrix that encodes the parent-child relationships defined by the sets $\mathcal{I}_i^{(l)}$. The entries of $\boldsymbol{S}_l$ are given by:

$$(\boldsymbol{S}_l)_{i,j} = \begin{cases} 1 & \text{if } j \in \mathcal{I}_i^{(l)} \\ 0 & \text{otherwise} \end{cases} \quad \text{for } i \in \mathcal{I}^{(l)}, j \in \mathcal{I}^{(l+1)} \tag{2.18}$$

Then, the aggregation constraint between adjacent levels can be written in matrix form as:

$$\hat{\boldsymbol{y}}_t^{(l)} = \boldsymbol{S}_l \hat{\boldsymbol{y}}_t^{(l+1)} \tag{2.19}$$

Remark (Properties of Summing Matrices). Given a summing matrix $\boldsymbol{S}_l \in \{0,1\}^{m_l \times m_{l+1}}$ between levels l and $l+1$, it has several important properties:

- We read the summing matrix as follows:
  - Rows: Each row $i \in \mathcal{I}^{(l)}$ in the summing matrix corresponds to a node at level l (parent level), where $\mathcal{I}^{(l)}$ is the set of all nodes at level l, with $|\mathcal{I}^{(l)}| = m_l$ rows in total
  - Columns: Each column $j \in \mathcal{I}^{(l+1)}$ in the summing matrix corresponds to a node at level $l+1$ (child level), where $\mathcal{I}^{(l+1)} = \bigcup_{i \in \mathcal{I}^{(l)}} \mathcal{I}_i^{(l)}$ represents all nodes at level $l+1$, with $|\mathcal{I}^{(l+1)}| = m_{l+1}$ columns in total
  - Entries: $(\boldsymbol{S}_l)_{i,j} = 1$ if and only if $j \in \mathcal{I}_i^{(l)}$, which directly reflects the hierarchical relationships encoded in the sets $\mathcal{I}_i^{(l)}$.
- Each column of $\boldsymbol{S}_l$ contains exactly one non-zero entry (equal to 1), as each node at level $l+1$ has exactly one parent at level l
- The row sum $\sum_{j=1}^{m_{l+1}} (\boldsymbol{S}_l)_{i,j}$ equals $|\mathcal{I}_i^{(l)}|$, representing the number of children of node i
- For levels $k > l$, the matrix product $\boldsymbol{S}_l \boldsymbol{S}_{l+1} \cdots \boldsymbol{S}_{k-1}$ represents the aggregation relationship between levels l and k

Example 2.5.7 (Matrix Representation of the Three-Level Hierarchy). Continuing with our company example, we can now express the hierarchical relationships using summing matrices. For this structure with $L = 3$, we need two summing matrices $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$. Following our previous indexing where $m_1 = 1$ (company level), $m_2 = 2$ (regional level), and $m_3 = 5$ (store level), these matrices are:

$$\boldsymbol{S}_1 = \begin{bmatrix} 1 & 1 \end{bmatrix} \in \{0,1\}^{1 \times 2} \quad \text{(company-to-region aggregation)} \tag{2.20}$$

$$\boldsymbol{S}_2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \in \{0,1\}^{2 \times 5} \quad \text{(region-to-store aggregation)} \tag{2.21}$$

These matrices exhibit several key properties:

1. Column uniqueness: In $\boldsymbol{S}_2$, each column contains exactly one 1, confirming our earlier observation that each store belongs to exactly one region. For example, the first column $[1\ 0]^T$ shows that store 1 belongs to the first region.
2. Row sums reflect the number of children each node has:
   - For Region 1: $\sum_{j=1}^{5} (\boldsymbol{S}_2)_{1,j} = 3 = |\mathcal{I}_1^{(2)}|$ (three stores)
   - For Region 2: $\sum_{j=1}^{5} (\boldsymbol{S}_2)_{2,j} = 2 = |\mathcal{I}_2^{(2)}|$ (two stores)
3. Multi-level aggregation: The relationship between company and store levels can be expressed through matrix multiplication:
   $$\boldsymbol{S}_1 \boldsymbol{S}_2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} \in \{0,1\}^{1 \times 5} \tag{2.22}$$
   showing how all stores contribute to the company total.

Using these matrices, we can write the coherency conditions compactly as:

$$\begin{aligned} \hat{\boldsymbol{y}}_t^{(1)} &= \boldsymbol{S}_1 \hat{\boldsymbol{y}}_t^{(2)} = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \hat{y}_{1,t}^{(2)} \\ \hat{y}_{2,t}^{(2)} \end{bmatrix} && \text{(company = sum of regions)} \\ \hat{\boldsymbol{y}}_t^{(2)} &= \boldsymbol{S}_2 \hat{\boldsymbol{y}}_t^{(3)} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} \hat{y}_{1,t}^{(3)} \\ \hat{y}_{2,t}^{(3)} \\ \hat{y}_{3,t}^{(3)} \\ \hat{y}_{4,t}^{(3)} \\ \hat{y}_{5,t}^{(3)} \end{bmatrix} && \text{(regions = sum of stores)} \end{aligned} \tag{2.23}$$

The direct relationship between company and store levels is then:

$$\hat{\boldsymbol{y}}_t^{(1)} = \boldsymbol{S}_1\boldsymbol{S}_2\hat{\boldsymbol{y}}_t^{(3)} = \begin{bmatrix}1 & 1 & 1 & 1 & 1\end{bmatrix}\hat{\boldsymbol{y}}_t^{(3)} \tag{2.24}$$

This matrix formulation provides a compact representation of the hierarchical relationships we previously expressed using sets and summations, and matches our earlier result in equation (2.12). It demonstrates how matrix multiplication naturally implements the aggregation of forecasts across multiple levels of the hierarchy.

Looking back at our three-level example, we can observe that the relationship between levels 1 and 3 can be obtained either directly through $\boldsymbol{S}_1\boldsymbol{S}_2$ or by computing each step separately and then combining:

$$\begin{bmatrix}1 & 1 & 1 & 1 & 1\end{bmatrix} = \begin{bmatrix}1 & 1\end{bmatrix}\begin{bmatrix}1 & 1 & 1 & 0 & 0\\ 0 & 0 & 0 & 1 & 1\end{bmatrix} \tag{2.25}$$

This suggests a general principle: the summing matrix between any two levels in a hierarchy can be expressed as a product of summing matrices between consecutive levels. This observation is not just a computational convenience, it reflects the fundamental nature of how aggregation works across multiple levels of the hierarchy. We can formalize this insight through the following definition and associated properties.

Definition 2.5.8 (Multi-level Summing Matrix)**.** For any two levels l, k where $l < k \leq L$, the summing matrix $\boldsymbol{S}_{l,k}$ that defines the relationship between forecasts at level l and level k is given by the product of consecutive single-level summing matrices:

$$\boldsymbol{S}_{l,k} = \prod_{i=l}^{k-1}\boldsymbol{S}_i \tag{2.26}$$

This matrix satisfies the aggregation relationship:

$$\hat{\boldsymbol{y}}_t^{(l)} = \boldsymbol{S}_{l,k}\hat{\boldsymbol{y}}_t^{(k)} \tag{2.27}$$

Proposition 2.5.9 (Properties of Multi-level Summing Matrices)**.** The multi-level summing matrices satisfy the following properties:

1. If $k = l$, then $\boldsymbol{S}_{l,l} = \boldsymbol{I}_{m_l}$, where $\boldsymbol{I}_{m_l}$ is the $m_l \times m_l$ identity matrix. If $k = l + 1$, then $\boldsymbol{S}_{l,l+1} = \boldsymbol{S}_l$.
2. For any three levels $l \leq m \leq k$, the matrices satisfy the composition rule $\boldsymbol{S}_{l,k} = \boldsymbol{S}_{l,m}\boldsymbol{S}_{m,k}$.
3. Each column of $\boldsymbol{S}_{l,k}$ contains exactly one non-zero entry.
4. The row sum of row i in $\boldsymbol{S}_{l,k}$ equals the total number of descendants of node i at level k.

Proof. For the first property, consider $k = l$. By definition, $\boldsymbol{S}_{l,l} = \prod_{i=l}^{l-1}\boldsymbol{S}_i$ is an empty product, which for matrices is defined as the identity matrix of appropriate size. Hence, $\boldsymbol{S}_{l,l} = \boldsymbol{I}_{m_l}$, consistent with the fact that each node is its own and only ancestor at the same level. When $k = l + 1$, we obtain $\boldsymbol{S}_{l,l+1} = \prod_{i=l}^{l}\boldsymbol{S}_i = \boldsymbol{S}_l$, as shown previously.

For the second property, take three levels $l \leq m \leq k$. Splitting the matrix product gives

$$\boldsymbol{S}_{l,k} = \prod_{i=l}^{k-1}\boldsymbol{S}_i = \left(\prod_{i=l}^{m-1}\boldsymbol{S}_i\right)\left(\prod_{i=m}^{k-1}\boldsymbol{S}_i\right) = \boldsymbol{S}_{l,m}\boldsymbol{S}_{m,k},$$

which establishes the composition rule $\boldsymbol{S}_{l,k} = \boldsymbol{S}_{l,m}\boldsymbol{S}_{m,k}$.

The third property can be proven by induction on the number of levels between l and k. For the base case $k = l + 1$, we have $\boldsymbol{S}_{l,k} = \boldsymbol{S}_l$, which by definition has exactly one 1 in each column. Assume the property holds for $\boldsymbol{S}_{l,k-1}$. Since $\boldsymbol{S}_{k-1,k} = \boldsymbol{S}_{k-1}$ also has exactly one 1 in each column, consider the matrix product $\boldsymbol{S}_{l,k} = \boldsymbol{S}_{l,k-1}\boldsymbol{S}_{k-1}$. For any column j in $\boldsymbol{S}_{l,k}$ (corresponding to node j at level k), $\boldsymbol{S}_{k-1}$ has exactly one row r^* such that $(\boldsymbol{S}_{k-1})_{r^*,j} = 1$ and $(\boldsymbol{S}_{k-1})_{r,j} = 0$ for all $r \neq r^*$. By the

inductive hypothesis, $\boldsymbol{S}_{l,k-1}$ has exactly one 1 in column r^*, located in row i^* with $(\boldsymbol{S}_{l,k-1})_{i^*,r^*} = 1$ and $(\boldsymbol{S}_{l,k-1})_{i,r^*} = 0$ for all $i \neq i^*$. Therefore, for the matrix product

$$(\boldsymbol{S}_{l,k})_{i,j} = \sum_{r=1}^{m_{k-1}} (\boldsymbol{S}_{l,k-1})_{i,r}(\boldsymbol{S}_{k-1})_{r,j} = (\boldsymbol{S}_{l,k-1})_{i,r^*},$$

since $(\boldsymbol{S}_{k-1})_{r,j} = 0$ for all $r \neq r^*$. It follows that $(\boldsymbol{S}_{l,k})_{i^*,j} = 1$ and $(\boldsymbol{S}_{l,k})_{i,j} = 0$ for all $i \neq i^*$. Thus, each column j of $\boldsymbol{S}_{l,k}$ contains exactly one 1, located in the row corresponding to the unique ancestor i^* of node j at level l, completing the induction.

For the fourth property, we show that $\sum_j (\boldsymbol{S}_{l,k})_{i,j}$ equals the number of descendants of node i at level k. For adjacent levels, the row sum $\sum_j (\boldsymbol{S}_l)_{i,j}$ equals $|\mathcal{I}_i^{(l)}|$, the number of children of node i. For multiple levels, we use Property 2 and write

$$(\boldsymbol{S}_{l,k})_{i,j} = \sum_{r=1}^{m_m} (\boldsymbol{S}_{l,m})_{i,r}(\boldsymbol{S}_{m,k})_{r,j}.$$

Summing over j counts all paths from node i at level l to any node at level k through the hierarchy. By the unique parent property (Property 3), each node at level k contributes exactly once to this sum if and only if it is a descendant of node i. Therefore, the row sum equals the total number of descendants of node i at level k. □

Remark. The properties established in the proposition above are purely structural and hold independently of whether the forecasts in the hierarchy are coherent. These matrices encode the hierarchical relationships between nodes, while coherency concerns the forecasts themselves. Specifically, Property 1 ($\boldsymbol{S}_{l,l} = \boldsymbol{I}_{m_l}$) expresses that a node is its own ancestor; Property 2 ($\boldsymbol{S}_{l,k} = \boldsymbol{S}_{l,m}\boldsymbol{S}_{m,k}$) reflects compositional connectivity between levels; Property 3 states that each node has a unique parent; and Property 4 shows that each row sum counts the number of descendants of a node at a deeper level. Coherency, on the other hand, requires that forecasts at different levels respect this fixed structure, i.e.,

$$\hat{\boldsymbol{y}}_t^{(l)} = \boldsymbol{S}_l \hat{\boldsymbol{y}}_t^{(l+1)}.$$

This condition may or may not hold for a given forecast, but it does not alter the structural properties of the summing matrices themselves. Once these properties are established, coherency can be understood as the consistency of forecasts across the hierarchy induced by these matrices.

Definition 2.5.10 (Stacked Representation)**.** Let $N = \sum_{l=1}^{L} m_l$ be the total number of nodes in the hierarchy. We define the stacking operator:

$$\text{Vec} : \mathcal{X} = \left(\bigtimes_{l=1}^{L} \mathbb{R}^{m_l} \right) \to \mathbb{R}^N$$

For any set of forecasts $\{\hat{\boldsymbol{y}}_t^{(l)}\}_{l=1}^{L}$, we define the stacked forecast vector:

$$\hat{\boldsymbol{y}}_t = \text{Vec}(\{\hat{\boldsymbol{y}}_t^{(l)}\}_{l=1}^{L}) = \begin{bmatrix} \hat{\boldsymbol{y}}_t^{(1)} \\ \hat{\boldsymbol{y}}_t^{(2)} \\ \vdots \\ \hat{\boldsymbol{y}}_t^{(L)} \end{bmatrix} \tag{2.28}$$

Similarly, we define the stacked summing matrix $S \in \mathbb{R}^{N \times m_L}$ as:

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{S}_{1,L} \\ \boldsymbol{S}_{2,L} \\ \vdots \\ \boldsymbol{I}_{m_L} \end{bmatrix} \tag{2.29}$$

Theorem 2.5.11 (Unified Coherency)**.** The following statements are equivalent:

1. The forecasts $\{\hat{\boldsymbol{y}}_t^{(l)}\}_{l=1}^{L}$ are coherent
2. $\hat{\boldsymbol{y}}_t^{(l)} = \boldsymbol{S}_l \hat{\boldsymbol{y}}_t^{(l+1)}$ for all $l \in \{1, \dots, L-1\}$
3. $\hat{\boldsymbol{y}}_t = \boldsymbol{S}\hat{\boldsymbol{y}}_t^{(L)}$

Proof. We show that (1) $\iff$ (2) and (2) $\iff$ (3):
(1) $\iff$ (2): This is immediate from the definition of coherency.
(2) $\implies$ (3): Assume $\hat{\boldsymbol{y}}_t^{(l)} = \boldsymbol{S}_l \hat{\boldsymbol{y}}_t^{(l+1)}$ holds for all l. Then by recursive substitution:

$$\hat{\boldsymbol{y}}_t^{(l)} = \boldsymbol{S}_l \hat{\boldsymbol{y}}_t^{(l+1)} = \boldsymbol{S}_l(\boldsymbol{S}_{l+1}\hat{\boldsymbol{y}}_t^{(l+2)}) = \cdots = (\prod_{i=l}^{L-1} \boldsymbol{S}_i)\hat{\boldsymbol{y}}_t^{(L)} = \boldsymbol{S}_{l,L}\hat{\boldsymbol{y}}_t^{(L)}.$$

This holds for each level l, therefore:

$$\hat{\boldsymbol{y}}_t = \begin{bmatrix} \hat{\boldsymbol{y}}_t^{(1)} \\ \hat{\boldsymbol{y}}_t^{(2)} \\ \vdots \\ \hat{\boldsymbol{y}}_t^{(L)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{S}_{1,L} \\ \boldsymbol{S}_{2,L} \\ \vdots \\ \boldsymbol{I}_{m_L} \end{bmatrix} \hat{\boldsymbol{y}}_t^{(L)} = \boldsymbol{S}\hat{\boldsymbol{y}}_t^{(L)}$$

(3) $\implies$ (2): Assume $\hat{\boldsymbol{y}}_t = \boldsymbol{S}\hat{\boldsymbol{y}}_t^{(L)}$. By the structure of $\boldsymbol{S}$, this means that for each level l:

$$\hat{\boldsymbol{y}}_t^{(l)} = \boldsymbol{S}_{l,L}\hat{\boldsymbol{y}}_t^{(L)}$$

Using the property $\boldsymbol{S}_{l,L} = \boldsymbol{S}_{l,l+1}\boldsymbol{S}_{l+1,L} = \boldsymbol{S}_l\boldsymbol{S}_{l+1,L}$ from our earlier results:

$$\begin{aligned} \hat{\boldsymbol{y}}_t^{(l)} &= \boldsymbol{S}_{l,L}\hat{\boldsymbol{y}}_t^{(L)} \\ &= \boldsymbol{S}_l\boldsymbol{S}_{l+1,L}\hat{\boldsymbol{y}}_t^{(L)} \\ &= \boldsymbol{S}_l\hat{\boldsymbol{y}}_t^{(l+1)} \end{aligned}$$

This holds for all $l \in \{1, \dots, L-1\}$, proving coherency. □

Proposition 2.5.12 (Equivalent Forms of Coherency)**.** For a hierarchical structure with L levels, let $\boldsymbol{S}_l \in \mathbb{R}^{m_l \times m_{l+1}}$ be the summing matrix between adjacent levels. For any levels l, k with $k > l$, define:

$$\boldsymbol{S}_{l,k} = \prod_{i=l}^{k-1} \boldsymbol{S}_i$$

Let $N = \sum_{l=1}^{L} m_l$ be the total number of nodes and define the stacked summing matrix $\boldsymbol{S} \in \mathbb{R}^{N \times m_L}$ as:

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{S}_{1,L} \\ \boldsymbol{S}_{2,L} \\ \vdots \\ \boldsymbol{S}_{L,L} \end{bmatrix}$$

Then for any set of forecasts $\{\hat{\boldsymbol{y}}_t^{(l)}\}_{l=1}^{L}$, the following statements are equivalent:

(i) The forecasts are coherent (i.e., $\hat{\boldsymbol{y}}_t^{(l)} = \boldsymbol{S}_l \hat{\boldsymbol{y}}_t^{(l+1)}$ for all $l \in \{1, \dots, L-1\}$)
(ii) $\hat{\boldsymbol{y}}_t^{(l)} = \boldsymbol{S}_{l,k}\hat{\boldsymbol{y}}_t^{(k)}$ for each $k > l$
(iii) $\hat{\boldsymbol{y}}_t^{(l)} = \boldsymbol{S}_{l,L}\hat{\boldsymbol{y}}_t^{(L)}$ for each l
(iv) $\text{Vec}(\hat{\boldsymbol{y}}_t) = \boldsymbol{S}\hat{\boldsymbol{y}}_t^{(L)}$

Proof. By the Unified Coherency theorem, we know that (i) $\iff$ Vec$(\hat{\boldsymbol{y}}_t) = \boldsymbol{S}\hat{\boldsymbol{y}}_t^{(L)} \iff$ (iv). Therefore, we only need to show that these are also equivalent to (ii) and (iii).

First, observe that (ii) $\implies$ (iii) is immediate by taking $k = L$.

Next, (iii) $\implies$ (iv) follows directly from the definition of Vec and $\boldsymbol{S}$:

$$\text{Vec}(\hat{\boldsymbol{y}}_t) = \begin{bmatrix} \hat{\boldsymbol{y}}_t^{(1)} \\ \hat{\boldsymbol{y}}_t^{(2)} \\ \vdots \\ \hat{\boldsymbol{y}}_t^{(L)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{S}_{1,L}\hat{\boldsymbol{y}}_t^{(L)} \\ \boldsymbol{S}_{2,L}\hat{\boldsymbol{y}}_t^{(L)} \\ \vdots \\ \boldsymbol{S}_{L,L}\hat{\boldsymbol{y}}_t^{(L)} \end{bmatrix} = \boldsymbol{S}\hat{\boldsymbol{y}}_t^{(L)}$$

Finally, to show (i) $\implies$ (ii), if the forecasts are coherent, then for any $k > l$:

$$\hat{\boldsymbol{y}}_t^{(l)} = \boldsymbol{S}_l\hat{\boldsymbol{y}}_t^{(l+1)} = \boldsymbol{S}_l(\boldsymbol{S}_{l+1}\hat{\boldsymbol{y}}_t^{(l+2)}) = \cdots = (\prod_{i=l}^{k-1}\boldsymbol{S}_i)\hat{\boldsymbol{y}}_t^{(k)} = \boldsymbol{S}_{l,k}\hat{\boldsymbol{y}}_t^{(k)}$$

The chain of implications (i) $\implies$ (ii) $\implies$ (iii) $\implies$ (iv) $\implies$ (i) completes the proof. □

The progression from recursive coherency to unified coherency, culminating in the stacked representation, provides a complete characterization of hierarchical forecasting structures. Through these results, we have established several fundamental properties:

1. The stacked representation $\hat{\boldsymbol{y}}_t = \boldsymbol{S}\hat{\boldsymbol{y}}_t^{(L)}$ captures all coherency constraints in a single equation, replacing the need for multiple pairwise relationships between levels. This not only simplifies notation but also provides a practical tool for verifying coherency across the entire hierarchy at once.
2. The summing matrix $\boldsymbol{S}$ completely characterizes the hierarchical structure. Its construction from individual level matrices $\boldsymbol{S}_l$ preserves all parent-child relationships, while its block structure reveals how forecasts at each level relate to the bottom-level forecasts.
3. Any coherent forecast can be generated from some bottom-level forecast $\hat{\boldsymbol{y}}_t^{(L)}$. This is not just a mathematical property but has practical implications: once we have bottom-level forecasts, we can generate coherent forecasts for the entire hierarchy through a single matrix multiplication.
4. The dimension of $\boldsymbol{S}$ ($N \times m_L$) represents the minimal representation needed to ensure coherency, where $N = 1 + \sum_{l=1}^{L} m_l$ is the total number of nodes in the hierarchy. This optimality in representation becomes particularly important when working with large-scale hierarchical structures.

These results form the foundation for developing reconciliation methods, which we will explore in subsequent sections. Whether working with bottom-up approaches, top-down methods, or optimal reconciliation techniques, the principles established here provide the mathematical framework for ensuring forecast coherency across hierarchical structures.

2.6. Forecasting Reconciliation

The stacked representation $\hat{\boldsymbol{y}}_t = \boldsymbol{S}\hat{\boldsymbol{y}}_t^{(L)}$ provides a complete characterization of coherent forecasts. However, in practice, independently generated forecasts at different levels rarely satisfy this coherency property. Forecast reconciliation addresses this issue by adjusting these independently generated forecasts to ensure coherency while maintaining or even improving accuracy.

2.6.1. From Coherency to Reconciliation

Consider independently generated forecasts $\hat{\boldsymbol{y}}_t \in \mathbb{R}^N$ for all nodes in the hierarchy. By our previous results, these forecasts are coherent if and only if $\hat{\boldsymbol{y}}_t = \boldsymbol{S}\hat{\boldsymbol{y}}_t^{(L)}$ for some bottom-level forecasts $\hat{\boldsymbol{y}}_t^{(L)}$. When this condition fails, we need a systematic approach to modify these forecasts to achieve coherency.

Definition 2.6.1 (Base Forecasts and Reconciliation)**.** Let $\hat{\boldsymbol{y}}_t \in \mathbb{R}^N$ be a vector of independently generated forecasts for all nodes in the hierarchy, which we call base forecasts. A reconciliation method involves three spaces:

- The full hierarchical space $\mathbb{R}^N$ containing forecasts for all nodes
- A reconciliation space $\mathbb{R}^{m_L}$ where coherency constraints are resolved
- The coherent hierarchical space (a subspace of $\mathbb{R}^N$) containing valid coherent forecasts

The method transforms base forecasts into coherent forecasts $\tilde{\boldsymbol{y}}_t$ through:

$$\tilde{\boldsymbol{y}}_t = \boldsymbol{S}\boldsymbol{P}\hat{\boldsymbol{y}}_t \tag{2.30}$$

where $\boldsymbol{P} \in \mathbb{R}^{m_L \times N}$ maps to the reconciliation space and $\boldsymbol{S} \in \mathbb{R}^{N \times m_L}$ maps back to the coherent hierarchical space.

The form of $\tilde{\boldsymbol{y}}_t = \boldsymbol{S}\boldsymbol{P}\hat{\boldsymbol{y}}_t$ ensures coherency by construction through the composition of the summing matrix $\boldsymbol{S}$ and reconciliation matrix $\boldsymbol{P}$. Starting with forecasts across all levels ($\hat{\boldsymbol{y}}_t \in \mathbb{R}^N$), the reconciliation matrix $\boldsymbol{P}$ maps these to a single level representation ($\boldsymbol{P}\hat{\boldsymbol{y}}_t \in \mathbb{R}^{m_L}$). The summing matrix $\boldsymbol{S}$ then maps this representation back to a complete hierarchical structure ($\tilde{\boldsymbol{y}}_t \in \mathbb{R}^N$), ensuring coherency through its encoding of the hierarchical relationships. Different choices of $\boldsymbol{P}$ lead to different reconciliation methods, which we explore next:

2.6.2. Bottom-Up Reconciliation

The simplest approach follows directly from our stacked representation: if we trust only the bottom-level forecasts, we can ignore all higher-level forecasts and aggregate upward:

Definition 2.6.2 (Bottom-Up Reconciliation Matrix)**.** The bottom-up approach sets:

$$\boldsymbol{P}_{BU} = \begin{bmatrix} \boldsymbol{0}_{m_L \times (N - m_L)} & \boldsymbol{I}_{m_L} \end{bmatrix} \tag{2.31}$$

which simply selects the bottom-level forecasts from $\hat{\boldsymbol{y}}_t$ and lets $\boldsymbol{S}$ handle the aggregation.

Corollary 2.6.3 (Coherency of Bottom-Up Reconciliation)**.** The bottom-up reconciled forecasts $\tilde{\boldsymbol{y}}_t = \boldsymbol{S}\boldsymbol{P}_{BU}\hat{\boldsymbol{y}}_t$ are coherent.

Proof. Let $\hat{\boldsymbol{y}}_t$ be any base forecasts. The bottom-up reconciliation matrix $\boldsymbol{P}_{BU}$ extracts the bottom-level forecasts:

$$\boldsymbol{P}_{BU}\hat{\boldsymbol{y}}_t = \begin{bmatrix} \boldsymbol{0}_{m_L \times (N - m_L)} & \boldsymbol{I}_{m_L} \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{y}}_t^{(1)} \\ \vdots \\ \hat{\boldsymbol{y}}_t^{(L-1)} \\ \hat{\boldsymbol{y}}_t^{(L)} \end{bmatrix} = \hat{\boldsymbol{y}}_t^{(L)} \tag{2.32}$$

Therefore:

$$\tilde{\boldsymbol{y}}_t = \boldsymbol{S}\boldsymbol{P}_{BU}\hat{\boldsymbol{y}}_t = \boldsymbol{S}\hat{\boldsymbol{y}}_t^{(L)} \tag{2.33}$$

By the Unified Coherency theorem, since $\tilde{\boldsymbol{y}}_t = \boldsymbol{S}\hat{\boldsymbol{y}}_t^{(L)}$, the bottom-up reconciled forecasts are coherent. □

This shows that bottom-up reconciliation is perhaps the most natural reconciliation method, as it directly implements the stacked representation form $\tilde{\boldsymbol{y}}_t = \boldsymbol{S}\hat{\boldsymbol{y}}_t^{(L)}$ and achieves coherency through the simplest possible choice of reconciliation matrix $\boldsymbol{P}$.

2.6.3. Top-Down Reconciliation

In contrast to bottom-up approaches, top-down methods trust the top-level forecast most and distribute it to lower levels using historical proportions. These proportions determine how much of the total forecast each bottom-level unit should receive.

Definition 2.6.4 (Top-Down Reconciliation Matrix)**.** The top-down approach sets:

$$\boldsymbol{P}_{TD} = \boldsymbol{p} \cdot \boldsymbol{e}_1^T \in \mathbb{R}^{m_L \times N} \tag{2.34}$$

where:

- $\boldsymbol{p} \in \mathbb{R}^{m_L}$ is a vector of proportions that satisfies $\sum_{i=1}^{m_L} p_i = 1$

- $\boldsymbol{e}_1 \in \mathbb{R}^N$ is the first standard basis vector (i.e., $[1, 0, \ldots, 0]^T$)

Corollary 2.6.5 (Coherency of Top-Down Reconciliation)**.** The top-down reconciled forecasts $\tilde{\boldsymbol{y}}_t = \boldsymbol{S}\boldsymbol{P}_{TD}\hat{\boldsymbol{y}}_t$ are coherent.

Proof. Let $\hat{\boldsymbol{y}}_t$ be any base forecasts. The top-down reconciliation matrix $\boldsymbol{P}_{TD}$ first produces candidate bottom-level forecasts:

$$\hat{\boldsymbol{y}}_t^{(L)} = \boldsymbol{P}_{TD}\hat{\boldsymbol{y}}_t = (\boldsymbol{p} \cdot \boldsymbol{e}_1^T)\hat{\boldsymbol{y}}_t = \boldsymbol{p}\hat{\boldsymbol{y}}_t^{(1)} \tag{2.35}$$

These candidate forecasts have two important properties:

- They sum to the top-level forecast: $\sum_{i=1}^{m_L} \hat{y}_{i,t}^{(L)} = \hat{\boldsymbol{y}}_t^{(1)}$ since $\sum_{i=1}^{m_L} p_i = 1$
- They are constructed as proportions of the top-level forecast: $\hat{y}_{i,t}^{(L)} = p_i\hat{\boldsymbol{y}}_t^{(1)}$

The reconciled forecasts are then obtained by:

$$\tilde{\boldsymbol{y}}_t = \boldsymbol{S}\boldsymbol{P}_{TD}\hat{\boldsymbol{y}}_t = \boldsymbol{S}\hat{\boldsymbol{y}}_t^{(L)} \tag{2.36}$$

By the Unified Coherency theorem, the top-down reconciled forecasts are coherent. □

This shows that top-down reconciliation achieves coherency by working in the opposite direction of bottom-up: it uses only the top-level forecast $\hat{y}_t^{(1)}$ and proportionally distributes it downward. The critical question becomes how to choose the proportions p, leading to several common approaches:

- Average Historical Proportions: $p_i = \frac{1}{T}\sum_{t=1}^{T} \frac{y_{i,t}}{\sum_{j=1}^{m_L} y_{j,t}}$
- Proportions of Historical Averages: $p_i = \frac{\frac{1}{T}\sum_{t=1}^{T} y_{i,t}}{\sum_{j=1}^{m_L} \frac{1}{T}\sum_{t=1}^{T} y_{j,t}}$
- Forecast Proportions: $p_i = \frac{\hat{y}_{i,t}^{(L)}}{\sum_{j=1}^{m_L} \hat{y}_{j,t}^{(L)}}$

2.6.4. Middle-Out Reconciliation

Middle out reconciliation combines elements of both bottom-up and top-down approaches by selecting a middle level k as the base level. Coherent forecasts above the level k are obtained through bottom-up aggregation, whereas coherent forecasts below are obtained through top-down disaggregation.

Definition 2.6.6 (Middle-Out Reconciliation)**.** For a chosen middle level k where $1 < k < L$, the middle-out reconciliation matrix $\boldsymbol{G}_{MO} = \boldsymbol{P} \cdot \boldsymbol{E}_k$, where:

- $\boldsymbol{E}_k \in \mathbb{R}^{m_k \times N}$ is a selection matrix that extracts the forecasts at level k
- $\boldsymbol{P} \in \mathbb{R}^{m_L \times m_k}$ is a matrix of proportions that distributes level k forecasts to level L

The middle-out approach balances bottom-up and top-down methods and allows flexibility in choosing the reconciliation level. It works best when middle layers contain the most reliable forecasts.

However, it has drawbacks. Implementation is more complex than simpler approaches. Choosing the right middle level requires domain knowledge and testing. The method is not theoretically optimal, so other approaches may perform better in specific situations.

2.6.5. Optimal (MinT/OLS)

Definition 2.6.7 (MinT (Minimum Trace) Reconciliation)**.** Let $\hat{\boldsymbol{y}}_{t+h} \in \mathbb{R}^N$ be base (possibly incoherent) h-step-ahead forecasts, $\boldsymbol{S} \in \mathbb{R}^{N \times m_L}$ the stacked summing matrix, and $\boldsymbol{W}_h = \mathrm{Var}(\hat{\boldsymbol{\varepsilon}}_{t+h})$ the covariance of base forecast errors. Assuming unbiased base forecasts and the unbiasedness constraint $\boldsymbol{P}\boldsymbol{S} = \boldsymbol{I}_{m_L}$, the *MinT* reconciliation chooses

$$\boldsymbol{P}_{\mathsf{MinT}} = \left(\boldsymbol{S}^\top \boldsymbol{W}_h^{-1} \boldsymbol{S}\right)^{-1} \boldsymbol{S}^\top \boldsymbol{W}_h^{-1}, \qquad \tilde{\boldsymbol{y}}_{t+h} = \boldsymbol{S}\,\boldsymbol{P}_{\mathsf{MinT}}\,\hat{\boldsymbol{y}}_{t+h}.$$

This corresponds to a generalized least squares (GLS) projection of $\hat{\boldsymbol{y}}_{t+h}$ onto the coherent subspace $\{\boldsymbol{S}\boldsymbol{b} : \boldsymbol{b} \in \mathbb{R}^{m_L}\}$ weighted by $\boldsymbol{W}_h^{-1}$, yielding minimum-variance linear unbiased reconciled forecasts.

Definition 2.6.8 (OLS Reconciliation (Special Case of MinT))**.** If the base error covariance admits the bottom-up structure $\boldsymbol{W}_h = \boldsymbol{S}\boldsymbol{\Omega}_h\boldsymbol{S}^\top$ for some bottom-level covariance $\boldsymbol{\Omega}_h$, MinT simplifies to ordinary least squares (OLS):

$$\boldsymbol{P}_{\text{OLS}} = \left(\boldsymbol{S}^\top\boldsymbol{S}\right)^{-1}\boldsymbol{S}^\top, \qquad \tilde{\boldsymbol{y}}_{t+h} = \boldsymbol{S}\,\boldsymbol{P}_{\text{OLS}}\,\hat{\boldsymbol{y}}_{t+h}.$$

Remark. MinT satisfies minimum-variance linear unbiasedness among coherent linear mappings; OLS is its special case under the structural covariance assumption above. Detailed derivations, equivalences (GLS form), and variance properties appear in [11, 28]. Practical estimators for $\boldsymbol{W}_h$ (diagonal, shrinkage, structural) and their trade-offs are also discussed therein and in related follow-ups.

The MinT approach is often presented as a ready-to-use reconciliation formula, but its underlying principle is fundamentally statistical: it provides the minimum-variance unbiased estimator of coherent forecasts. Since the preceding sections have established the probabilistic and structural foundations of hierarchical forecasting, it is instructive to make explicit how MinT arises from these principles. The derivation below, following [28], shows how the reconciliation problem can be formulated as a constrained trace-minimization problem whose solution yields the familiar closed-form expression.

The reconciliation problem seeks a linear transformation $\tilde{\boldsymbol{y}}_{t+h} = \boldsymbol{S}\boldsymbol{P}\hat{\boldsymbol{y}}_{t+h}$ such that the reconciled forecasts are coherent, i.e., consistent with the aggregation constraints imposed by $\boldsymbol{S}$. Among all such linear mappings, we seek the one that minimizes the total variance of the reconciled forecast errors:

$$\min_{\boldsymbol{P}} \ \text{tr}\big(\text{Var}(\tilde{\boldsymbol{y}}_{t+h} - \boldsymbol{y}_{t+h})\big) \quad \text{s.t.} \quad \boldsymbol{P}\boldsymbol{S} = \boldsymbol{I}_{m_L}.$$

Assuming unbiased base forecasts, the forecast error covariance is $\text{Var}(\hat{\boldsymbol{y}}_{t+h} - \boldsymbol{y}_{t+h}) = \boldsymbol{W}_h$. Hence, the covariance of the reconciled errors is

$$\text{Var}(\tilde{\boldsymbol{y}}_{t+h} - \boldsymbol{y}_{t+h}) = \boldsymbol{S}\boldsymbol{P}\,\boldsymbol{W}_h\,\boldsymbol{P}^\top\boldsymbol{S}^\top.$$

The optimization problem becomes

$$\min_{\boldsymbol{P}} \ \text{tr}\big(\boldsymbol{S}\boldsymbol{P}\boldsymbol{W}_h\boldsymbol{P}^\top\boldsymbol{S}^\top\big) \quad \text{s.t.} \quad \boldsymbol{P}\boldsymbol{S} = \boldsymbol{I}_{m_L}.$$

Following [28], this problem can be simplified by introducing the transformed variables

$$\boldsymbol{L} = \boldsymbol{W}_h^{-1/2}\boldsymbol{S}, \qquad \boldsymbol{H} = \boldsymbol{P}\boldsymbol{W}_h^{1/2}.$$

Substituting into the objective yields an equivalent form:

$$\min_{\boldsymbol{H}} \ \text{tr}\big(\boldsymbol{H}\boldsymbol{H}^\top\big) \quad \text{s.t.} \quad \boldsymbol{H}\boldsymbol{L} = \boldsymbol{I}.$$

This is a constrained least squares problem with solution given by the Moore–Penrose pseudoinverse of $\boldsymbol{L}$:

$$\boldsymbol{H} = (\boldsymbol{L}^\top\boldsymbol{L})^{-1}\boldsymbol{L}^\top.$$

Reverting to the original variables, we obtain the optimal MinT reconciliation matrix:

$$\boldsymbol{P}_{\text{MinT}} = (\boldsymbol{S}^\top\boldsymbol{W}_h^{-1}\boldsymbol{S})^{-1}\boldsymbol{S}^\top\boldsymbol{W}_h^{-1}.$$

Finally, substituting $\boldsymbol{P}_{\text{MinT}}$ into $\tilde{\boldsymbol{y}}_{t+h} = \boldsymbol{S}\boldsymbol{P}\hat{\boldsymbol{y}}_{t+h}$ gives the reconciled forecasts:

$$\tilde{\boldsymbol{y}}_{t+h} = \boldsymbol{S}(\boldsymbol{S}^\top\boldsymbol{W}_h^{-1}\boldsymbol{S})^{-1}\boldsymbol{S}^\top\boldsymbol{W}_h^{-1}\hat{\boldsymbol{y}}_{t+h}.$$

When $\boldsymbol{W}_h$ is proportional to the identity, this reduces to the ordinary least squares (OLS) solution:

$$\boldsymbol{P}_{\text{OLS}} = (\boldsymbol{S}^\top\boldsymbol{S})^{-1}\boldsymbol{S}^\top.$$

Following the result of [11], the MinT estimator derived above is the *minimum-variance linear unbiased estimator* (MVLUE) among all coherent linear reconciliations.

Under the assumption that the base forecast errors have covariance $\text{Var}(\hat{\boldsymbol{y}}_{t+h} - \boldsymbol{y}_{t+h}) = \boldsymbol{W}_h = S\Omega_h S^\top$, the generalized least-squares solution

$$\boldsymbol{P}_{\text{MinT}} = (\boldsymbol{S}^\top\boldsymbol{W}_h^{-1}\boldsymbol{S})^{-1}\boldsymbol{S}^\top\boldsymbol{W}_h^{-1}$$

is equivalent to the best linear unbiased estimator of the bottom-level means in the regression formulation $\hat{\boldsymbol{y}}_{t+h} = S\boldsymbol{\beta} + \boldsymbol{\varepsilon}$. Consequently, the reconciled forecasts $\tilde{\boldsymbol{y}}_{t+h} = \boldsymbol{S}\boldsymbol{P}_{\text{MinT}}\hat{\boldsymbol{y}}_{t+h}$ achieve the smallest total forecast-error variance subject to unbiasedness [11].

2.6.6. Projection View of Reconciliation

All reconciliation methods can be interpreted as projections onto the coherent subspace.

Definition 2.6.9 (Orthogonal Projection onto the Coherent Subspace)**.** Let $\mathcal{S} = \{\boldsymbol{S}\boldsymbol{b} : \boldsymbol{b} \in \mathbb{R}^{m_L}\}$ denote the coherent subspace. The orthogonal projection onto $\mathcal{S}$ is

$$\boldsymbol{P}_{\mathcal{S}} = \boldsymbol{S}(\boldsymbol{S}^T\boldsymbol{S})^{-1}\boldsymbol{S}^T, \qquad \tilde{\boldsymbol{y}} = \boldsymbol{P}_{\mathcal{S}}\hat{\boldsymbol{y}}.$$

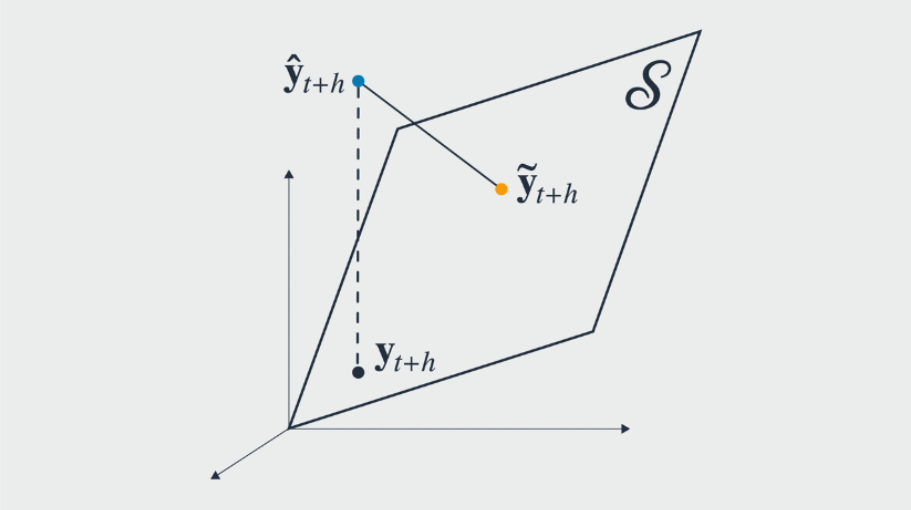


Figure 2.3: Geometric interpretation of forecast reconciliation as projection onto the coherent subspace. Adapted from Rangapuram et al. (2021) [25].

Figure 2.3 shows the intuition:

- The base forecast $\hat{\boldsymbol{y}}_{t+h}$ (blue) lies outside $\mathcal{S}$.
- The reconciled forecast $\tilde{\boldsymbol{y}}_{t+h}$ (orange) lies within $\mathcal{S}$ and satisfies all summation constraints.
- The actual value $\boldsymbol{y}_{t+h}$ (gray) is unknown and may lie anywhere.

Methods like Bottom-Up and Top-Down specify this projection through design rules, while OLS and MinT implement it statistically as least-squares projections—orthogonal for OLS, oblique (weighted by $\boldsymbol{W}_h^{-1}$) for MinT. This view clarifies that reconciliation enforces coherency by replacing the base forecast with its projection into the coherent subspace.

3

Graph Neural Networks for Demand Sensing

Chapter 2 formalized forecasting as learning unknown functions $g_i(\cdot)$ that map the past observations and covariates of each product i to its future demand. Those estimators were either trained independently per product (1ML-ForEach) or jointly on all products (1ML-ForAll).

While effective in many settings, these approaches ignore known relational structure between products. In hierarchical supply chains, demand patterns propagate along parent–child relations, as sibling products compete for shared market share, and sparse series can benefit from similar neighbors. To address this, we extend the multi-product demand process from Chapter 2 to incorporate an explicit graph structure.

Definition 3.0.1 (Graph-Based Multi-Product Demand Process)**.** Let $G = (V, E)$ be a graph where each node $i \in V$ represents a product with demand series $\{X_{i,t}\}$ and covariates $\{Z_{i,t}\}$. Define the L-lag neighborhood of node i as

$$\mathcal{N}^L(i) = \{(j, l) \mid j \in \mathcal{N}(i) \cup \{i\}, l = 1, \ldots, L\}, \tag{3.1}$$

$$\mathcal{N}(i) = \{j \in V : (i, j) \in E \text{ or } (j, i) \in E\}. \tag{3.2}$$

Then the demand process is

$$X_{i,t} = g_i\Big(\{(X_{j,t-l}, Z_{j,t-l}) : (j, l) \in \mathcal{N}^L(i)\},\ G\Big) + \varepsilon_{i,t}, \quad \forall i \in V. \tag{3.3}$$

This formulation lets each g_i condition only on a local neighborhood $\mathcal{N}(i)$, offering a middle ground between fully independent (1ML-ForEach) and fully pooled (1ML-ForAll) estimation. In the rest of this chapter, we present how Graph Neural Networks (GNNs) can be used to parameterize these graph-based demand processes $g_i(\cdot, G)$.

3.1. Theoretical Foundations for Graph Learning

We now review two key theoretical foundations that enable learning such functions: spectral graph theory, which underpins Graph Convolutional Networks (GCNs), and attention mechanisms, which underpins Graph Attention Networks (GATs). Together, these methods provide strategies for modeling cross-product relationships within hierarchical demand structures.

3.1.1. Spectral Graph Theory (Basis of GCNs)

Spectral graph theory provides the mathematical foundation for early graph neural networks by interpreting message passing as localized convolutions in the spectral domain. This view connects the structure of a graph to the propagation of information across its nodes.

Definition 3.1.1 (Graph Laplacian)**.** Let $G = (V, E)$ be an undirected graph with adjacency matrix $\boldsymbol{A} \in \{0,1\}^{n \times n}$ and degree matrix $\boldsymbol{D}$ defined by $D_{ii} = \sum_j A_{ij}$. The unnormalized graph Laplacian is

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}, \tag{3.4}$$

and the symmetrically normalized Laplacian is

$$\boldsymbol{L}_{\text{sym}} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}. \tag{3.5}$$

$\boldsymbol{L}$ is positive semi-definite and admits an eigendecomposition $\boldsymbol{L} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$, where $\boldsymbol{U}$ contains orthonormal eigenvectors and $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_n)$ the non-negative eigenvalues.

Definition 3.1.2 (Graph Fourier Transform)**.** Given a signal $\boldsymbol{x} \in \mathbb{R}^n$ on the nodes, its graph Fourier transform is

$$\hat{\boldsymbol{x}} = \boldsymbol{U}^\top \boldsymbol{x}, \qquad \boldsymbol{x} = \boldsymbol{U}\hat{\boldsymbol{x}}. \tag{3.6}$$

Here, eigenvalues λ_i represent graph frequencies: small λ_i correspond to smooth variations, large λ_i to rapid changes between neighbors. This enables defining graph convolutions as spectral filters.

Definition 3.1.3 (Spectral Graph Convolution)**.** Let $\boldsymbol{x} \in \mathbb{R}^n$ be a signal on the nodes of graph $G = (V, E)$ with Laplacian $\boldsymbol{L} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$. A spectral graph convolution with filter g_θ is defined as

$$\boldsymbol{y} = g_\theta(\boldsymbol{L})\boldsymbol{x} = \boldsymbol{U} g_\theta(\boldsymbol{\Lambda})\boldsymbol{U}^\top \boldsymbol{x}, \tag{3.7}$$

where $g_\theta(\boldsymbol{\Lambda}) = \text{diag}(g_\theta(\lambda_1), \ldots, g_\theta(\lambda_n))$ is a learnable diagonal filter in the spectral domain.

This formulation requires transforming signals to and from the spectral basis via $\boldsymbol{U}$, which involves dense matrix multiplications of order $\mathcal{O}(n^2)$ and an eigendecomposition of $\boldsymbol{L}$ with cost $\mathcal{O}(n^3)$. Such operations are infeasible for large graphs (e.g., thousands of products).

To avoid explicit eigenvector computations, Hammond, Vandergheynst, and Gribonval proposed approximating the spectral filter by a truncated expansion in Chebyshev polynomials:

$$g_\theta(\boldsymbol{L}) \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{\boldsymbol{L}}), \qquad \tilde{\boldsymbol{L}} = \frac{2\boldsymbol{L}}{\lambda_{\text{max}}} - \boldsymbol{I}, \tag{3.8}$$

where $T_k(\cdot)$ denotes the k-th Chebyshev polynomial defined recursively as

$$T_0(\boldsymbol{L}) = \boldsymbol{I}, \quad T_1(\boldsymbol{L}) = \boldsymbol{L}, \tag{3.9}$$

$$T_k(\boldsymbol{L}) = 2\boldsymbol{L}T_{k-1}(\boldsymbol{L}) - T_{k-2}(\boldsymbol{L}), \quad k \geq 2. \tag{3.10}$$

Chebyshev polynomials form an orthogonal basis on $[-1, 1]$ and provide numerically stable polynomial approximations of smooth functions. Since each term $T_k(\boldsymbol{L})\boldsymbol{x}$ depends only on nodes within k hops, the resulting filter is K-localized and can be computed without eigendecomposition.

Building on this idea, Kipf and Welling proposed a first-order ($K = 1$) approximation that yields the familiar Graph Convolutional Network (GCN) layer:

$$\boldsymbol{H}^{(l+1)} = \sigma\big(\hat{\boldsymbol{D}}^{-1/2}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-1/2}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}\big), \qquad \hat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}. \tag{3.11}$$

Adding self-loops via $\hat{\boldsymbol{A}}$ and symmetric normalization with $\hat{\boldsymbol{D}}$ stabilizes training and ensures that each node updates its representation as a normalized average of its own and its neighbors' features. In the demand forecasting context, this operation enables local information sharing between related products (e.g., SKUs in the same family) while keeping computation linear in the number of edges.

3.1.2. Attention Mechanisms (Basis of GATs)

While spectral methods like GCNs assign fixed weights to neighbors based solely on graph structure, many real-world graphs require context-dependent weighting. Attention mechanisms address this by allowing each node to learn how much to attend to each of its neighbors based on their current features. This forms the theoretical basis of Graph Attention Networks (GATs).

Definition 3.1.4 (Attention Mechanism)**.** Given a set of input vectors $\{\boldsymbol{h}_j\}_{j\in\mathcal{N}(i)}$, attention computes a weighted sum

$$\boldsymbol{h}'_i = \sum_{j\in\mathcal{N}(i)} \alpha_{ij}\boldsymbol{W}\boldsymbol{h}_j, \tag{3.12}$$

where α_{ij} are attention coefficients satisfying $\sum_{j\in\mathcal{N}(i)} \alpha_{ij} = 1$ and $\boldsymbol{W}$ is a learnable weight matrix.

These coefficients are obtained by measuring the relevance of neighbor j to target node i. A common implementation is the additive attention scoring function:

$$e_{ij} = \mathsf{LeakyReLU}\big(\boldsymbol{a}^\top[\boldsymbol{W}\boldsymbol{h}_i \,\|\, \boldsymbol{W}\boldsymbol{h}_j]\big), \tag{3.13}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}(i)} \exp(e_{ik})}, \tag{3.14}$$

where $\|$ denotes concatenation and $\boldsymbol{a}$ is a learnable parameter vector. This allows each node to dynamically assign higher weight to more relevant neighbors.

Definition 3.1.5 (Multi-Head Attention)**.** To improve stability and representational capacity, K independent attention heads are applied in parallel:

$$\boldsymbol{h}'_i = \|_{k=1}^{K}\sigma\Big(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(k)}\boldsymbol{W}^{(k)}\boldsymbol{h}_j\Big). \tag{3.15}$$

Each head learns a distinct relevance pattern, and concatenating them captures diverse types of interactions (e.g. seasonal similarity, price sensitivity, geographic proximity).

In GCNs, the influence of each neighbor j on node i is determined solely by the graph topology through the normalized adjacency coefficients $\hat{D}_{ii}^{-1/2}\hat{D}_{jj}^{-1/2}$ in the update rule $\boldsymbol{H}^{(l+1)} = \sigma(\hat{\boldsymbol{D}}^{-1/2}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-1/2}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)})$. By replacing these fixed structural weights with learnable attention coefficients α_{ij}, Graph Attention Networks generalize the GCN framework to a feature-dependent aggregation scheme.

Moreover, unlike in GCNs, the attention formulation can be extended to incorporate edge-level attributes. The attention score e_{ij} depends only on the features of nodes i and j. When additional edge information $\boldsymbol{e}_{ij}$ is available, such as categorical similarity, spatial distance, or hierarchical linkage strength, it can be integrated into the attention mechanism as

$$e_{ij} = \mathsf{LeakyReLU}\big(\boldsymbol{a}^\top[\,\boldsymbol{W}\boldsymbol{h}_i \,\|\, \boldsymbol{W}\boldsymbol{h}_j \,\|\, \boldsymbol{W}_e\boldsymbol{e}_{ij}\,]\big), \tag{3.16}$$

where $\boldsymbol{W}_e$ projects edge features into the same embedding space as node features. This ensures that edge attributes contribute on the same scale as node features within the attention scoring function.

3.2. Application to Hierarchical Time Series

The theoretical tools introduced above become practically useful once we map hierarchical time series into graph structures. This section explains how to (i) represent hierarchies as graphs, (ii) incorporate temporal information, and (iii) construct an end-to-end forecasting pipeline.

3.2.1. Hierarchical Structure as Graphs

Let the product hierarchy from Chapter 2 consist of levels $\mathcal{L} = \{1, \ldots, L\}$ with bottom level L and summing matrix $\boldsymbol{S}$. We represent this hierarchy as a graph $G = (V, E)$:

- Each node $i \in V$ corresponds to one time series $X_{i,t}$.
- Each edge $(i, j) \in E$ connects parent i to child j whenever $\boldsymbol{S}_{ij} = 1$.
- Additional lateral edges may connect siblings or geographically related nodes (optional).

Formally, for any node i at level l, its neighborhood is

$$\mathcal{N}(i) = \{j \mid (i, j) \in E \text{ or } (j, i) \in E\}, \tag{3.17}$$

which determines the message-passing scope during GNN training.

- *Node features* $\boldsymbol{x}_{i,t}$ are typically concatenated

$$\boldsymbol{x}_{i,t} = \big[X_{i,t-1:t-L},\ \text{day-of-week},\ \text{promo indicators},\ \text{weather},\ \dots\big], \tag{3.18}$$

  where Encode temporal patterns into the node feature vectors (e.g. 28-day lags, rolling means, seasonality flags). GNN layers then learn static cross-series patterns on these features.
- *Edge features* (only applicable with GATs or other edge-aware variants) may encode semantic similarity (e.g. category overlap) or spatial distance.

This graph representation embeds the full hierarchy directly into the model's input structure, allowing GNN layers to exchange information across parent–child and sibling nodes.

3.2.2. Forecasting Pipeline

Combining the above pieces yields an end-to-end graph-based forecasting pipeline:

1. **Graph construction** Build $G = (V, E)$ from the product hierarchy, add optional correlation-based edges, and compute node/edge features.
2. **Feature extraction** For each node i, construct $\boldsymbol{x}_{i,t}$ from historical lags, covariates, and exogenous drivers.
3. **GNN encoding** Apply L layers of GCN or GAT:

$$\boldsymbol{H}^{(l+1)} = \sigma\big(\text{GNNLayer}^{(l)}(\boldsymbol{H}^{(l)}, \boldsymbol{A})\big) \quad \text{with} \quad \boldsymbol{H}^{(0)} = \boldsymbol{X}_t. \tag{3.19}$$

4. **Forecast generation** Attach an MLP head to bottom-level nodes to output h-step forecasts $\hat{X}_{i,t+1:t+h}$.
5. **Reconciliation** There are three main strategies to enforce coherence in GNN-based forecasting:

   (a) Bottom-up prediction (hard coherence): Predict only bottom-level series with the GNN, and obtain all higher-level forecasts by summation:

$$\hat{\boldsymbol{y}} = \boldsymbol{S}\hat{\boldsymbol{b}}. \tag{3.20}$$

   This enforces coherence by construction. It also reduces the number of output heads, simplifying optimization. However, it restricts the model from directly using parent-level targets during training.

   (b) Coherence regularization (soft coherence): Predict all levels and add a penalty term to the loss:

$$\mathcal{L} = \mathcal{L}_{\text{forecast}} + \lambda_{\text{coh}} \big\| \hat{\boldsymbol{y}} - \boldsymbol{S}\hat{\boldsymbol{b}} \big\|_2^2. \tag{3.21}$$

   This encourages but does not force coherence. Choosing λ_{coh} balances accuracy vs. consistency.

   (c) Post-hoc reconciliation: Let the GNN produce incoherent forecasts $\hat{\boldsymbol{y}}$, then reconcile them afterward using optimal reconciliation (MinT/OLS, see Chapter 2):

$$\tilde{\boldsymbol{y}} = \boldsymbol{S}\boldsymbol{P}_{\text{MinT}}\hat{\boldsymbol{y}}. \tag{3.22}$$

   This fully enforces coherence but adds an extra step and assumes known forecast error covariances.

This pipeline integrates hierarchical structure and cross-series dependencies directly into the learning process, while preserving the coherence properties formalized in Chapter 2.

4

Data Application

This chapter outlines the application of the forecasting methodology described in the previous chapters, using open-source data from the M5 competition. M5 focused on a retail sales forecasting application with the objective to produce the most accurate point forecasts for 42,840 time series that represent the hierarchical unit sales of the largest retail company in the world, Walmart.

4.1. M5 dataset description

The M5 dataset, encompasses detailed unit sales data for various products sold across the United States. This dataset is structured as grouped time series, offering a view of sales trends and patterns. Specifically, the dataset includes unit sales data for 3,049 products, classified into three primary product categories: Hobbies, Foods, and Household. These categories are further divided into seven distinct product departments. The products are sold across ten different stores located in three states: California (CA), Texas (TX), and Wisconsin (WI). This hierarchical structure allows for analysis at multiple levels, including product-store unit sales, product categories, and geographical regions [17].

4.1.1. Dataset Composition

The M5 dataset consists of three main files:

Calendar.csv

This file provides detailed information about the dates on which the products were sold. It includes fields such as:

- `date`: The date in "y-m-" format.
- `wm_yr_wk`: The ID of the week the date belongs to.
- `weekday`: The type of the day (Saturday, Sunday, ..., Friday).
- `wday`: The ID of the weekday, starting from Saturday.
- `month`: The month of the date.
- `year`: The year of the date.
- `event_name_1`: Name of the first event on the date, if any.
- `event_type_1`: Type of the first event on the date, if any.
- `event_name_2`: Name of the second event on the date, if any.
- `event_type_2`: Type of the second event on the date, if any.
- `snap_CA`, `snap_TX`, `snap_WI`: Binary variables indicating whether SNAP purchases were allowed in stores in CA, TX, or WI on the given date.

Sell_prices.csv

This file contains information about the price of products sold per store and date, including:

- `store_id`: The ID of the store where the product was sold.

- `item_id`: The ID of the product.
- `wm_yr_wk`: The ID of the week.
- `sell_price`: The price of the product for the given week/store. If not available, it indicates that the product was not sold during that week.

Sales_train.csv
This file records the historical daily unit sales data per product and store. It includes:

- `item_id`: The ID of the product.
- `dept_id`: The ID of the department the product belongs to.
- `cat_id`: The ID of the category the product belongs to.
- `store_id`: The ID of the store where the product is sold.
- `state_id`: The state where the store is located.
- `d_1, d_2, ..., d_i, ..., d_1941`: The number of units sold on day `i`, starting from 2011-01-29.

4.1.2. Aggregation Levels

The M5 dataset's hierarchical structure allows for analysis at various aggregation levels, providing a flexible framework for different types of analysis. The levels of aggregation include:

- **Level 1**: Unit sales of all products, aggregated across all stores/states.
- **Level 2**: Unit sales of all products, aggregated for each state.
- **Level 3**: Unit sales of all products, aggregated for each store.
- **Level 4**: Unit sales of all products, aggregated for each category.
- **Level 5**: Unit sales of all products, aggregated for each department.
- **Level 6**: Unit sales of all products, aggregated for each state and category.
- **Level 7**: Unit sales of all products, aggregated for each state and department.
- **Level 8**: Unit sales of all products, aggregated for each store and category.
- **Level 9**: Unit sales of all products, aggregated for each store and department.
- **Level 10**: Unit sales of a specific product, aggregated across all stores/states.
- **Level 11**: Unit sales of a specific product, aggregated for each state.
- **Level 12**: Unit sales of a specific product, aggregated for each store.

4.1.3. Historical Data Range

The historical data in the M5 dataset spans from January 29, 2011, to June 19, 2016, providing a maximum selling history of 1,941 days or approximately 5.4 years. This extensive time range allows for robust time series analysis and forecasting [17].

4.1.4. Unique Features of the M5 Dataset

The M5 dataset stands out due to several unique features:

- **Grouped Unit Sales Data**: The dataset starts at the product-store level and is aggregated up to the levels of product departments, categories, stores, and geographical regions.
- **Explanatory Variables**: In addition to time series data, the dataset includes explanatory variables such as sell prices, promotions, days of the week, and special events like Super Bowl and Valentine's Day, which can influence unit sales and enhance forecasting accuracy.
- **Intermittency**: The dataset includes series that display intermittency, characterized by sporadic demand and frequent zeros, posing unique challenges for forecasting models [17].

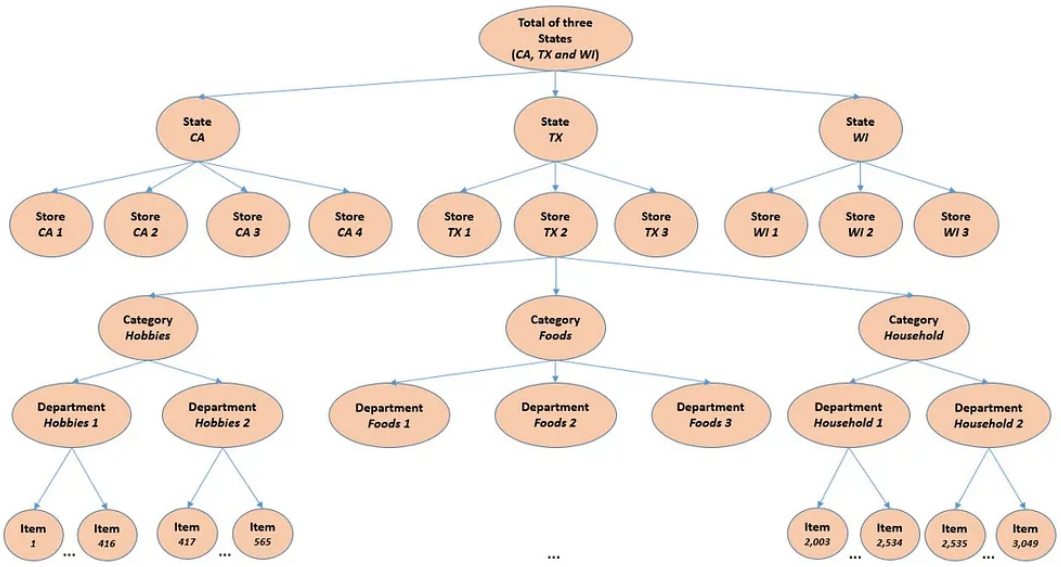


Figure 4.1: M5's hierarchy tree.

4.2. Setting up the Benchmark for the M5 Dataset

4.2.1. Load and Process the M5 Dataset

The initial step involves loading the M5 dataset, which consists of three files: `calendar.csv`, `sell_prices.csv`, and `sales_train.csv`. This process includes:

- Parsing each file to ensure accurate data types are maintained, especially for dates and categorical variables.
- Handling any missing or anomalous data that could affect the quality of forecasts.

4.2.2. Enrichment of Dataset Using External Drivers

Aside from working with the M5 data, an optional step is to enrich the dataset with external drivers that may influence product sales. These drivers include:

- Weather information separated for each state (from NOAA)
- Price information (from M5 dataset)
- Promotion information related to the US's SNAP program (from M5)
- Macroeconomic indicators (from FRED)
- Seasonality information (from M5)
- Events including holidays, cultural events, sporting events, religious events, and rare events such as natural disasters and eclipses (from M5 and FEMA)
- Location information, as product sales are segregated at store-level across different states (from M5)

It's worth noting that we were unable to incorporate inventory information and detailed product attributes as these were not available in the competition dataset.

4.2.3. Aggregation of Sales

To facilitate analysis at different levels, we aggregated sales data to match various hierarchies in the dataset, such as store-wise, state-wise, category-wise. This step is crucial for subsequent analysis and forecasting at different granularities.

4.2.4. Second Enrichment of Dataset

Following the initial sales aggregation, we enriched the dataset further by incorporating lagged sales data and rolling statistics like moving averages, which are vital for capturing trends and seasonality in time series data.

4.2.5. Preprocessing

Preprocessing the enriched dataset involves:

- Normalization or standardization of features to ensure that model inputs have similar scales.
- Encoding categorical variables using methods like one-hot encoding to convert them into a machine-readable format.
- Splitting the data into training and validation sets to facilitate the model training and tuning process.

In our specific implementation, we utilized the following train/validation/test split:

- Training set: 1,521 days
- Validation set: 322 days
- Test set: 98 days
- Total dataset: 1,941 days

This setup ensured sufficient historical data for model training while allowing for realistic validation and testing scenarios.

4.2.6. Fit & Tune Models

Software and implementation. All experiments were implemented in Python 3.12 using open-source libraries: `sktime` for pipeline composition and hierarchical reconciliation, `statsmodels` for statistical forecasting models, `scikit-learn` and `lightgbm` for tree-based machine learning methods, and `PyTorch` with `PyTorch Geometric` for deep learning and graph-based architectures. All runs were executed on an Apple M1 system (macOS) with 16 GB unified memory using a reproducible `conda` environment.

Training and validation. Models were trained on the first 1,521 days of the M5 dataset and validated using a rolling-origin (backtesting) procedure across the following 322 days, with a forecast horizon of $H{=}28$ days. Each validation fold advanced the forecast origin by 28 days, resulting in multiple overlapping evaluation windows. This setup mimics real-world forecasting conditions where models are retrained periodically as new data arrive.

Hyperparameter tuning via random search with backtesting. For each model, hyperparameters were tuned via random search, evaluated under the rolling-origin validation described above. Random search was implemented using `sklearn.model_selection.RandomizedSearchCV` for machine learning models, `sktime.forecasting.model_selection.ForecastingGridSearchCV` for statistical models, and custom training loops for neural networks. For each model, 30–50 random configurations were sampled, and the average RMSSE across all validation folds was used as the objective metric. The best-performing configuration was refit on the combined training and validation data before test evaluation.

Model classes and hyperparameters. The following model families and key hyperparameters were explored:

- **Statistical Models (baseline):** AutoARIMA (`statsmodels.tsa.arima.model.ARIMA`) with $p, d, q \in [0, 5]$, ETS (`sktime.forecasting.ets.ExponentialSmoothing`) with additive trend and seasonality, and TBATS (`sktime.forecasting.tbats.TBATS`) for complex seasonal patterns.
- **Machine Learning Models:** Random Forest (`sklearn.ensemble.RandomForestRegressor`) with n_estimators $\in [100, 500]$, max_depth $\in [5, 20]$; LightGBM (`lightgbm.LGBMRegressor`) with num_leaves $\in [2^6, 2^{11}]$, learning_rate $\in [10^{-3}, 10^{-1}]$ (log-uniform), subsample $\in [0.5, 1.0]$, feature_fraction $\in [0.5, 1.0]$, max_bin $\in [50, 150]$, and objective = `tweedie`.

- **Deep Learning Models:** LSTM (`torch.nn.LSTM`) with hidden size $\in [32, 256]$, layers $\in \{1, 2, 3\}$, dropout $\in [0, 0.3]$, and learning rate $\in [10^{-4}, 10^{-2}]$. Sequence length (input window) = 28, forecast horizon = 28.
- **Graph Neural Network Models:** Hierarchical Graph Network (HGN) implemented in `PyTorch Geometric` using spectral GCN layers with hidden dimension $\in [32, 128]$, Chebyshev polynomial order $K = 1$, and coherence regularization weight $\lambda_{\text{coh}} \in [0, 1]$. Optimization used Adam with early stopping on validation RMSSE.

Performance metrics. Model accuracy for tuning was optimized on RMSSE (per-fold) during backtesting. Final reporting follows the M5 protocol using normalized MAE (nMAE) and Weighted RMSSE (WRMSSE) on the test window.

4.2.7. Predict & Reconcile

After hyperparameter optimization, all models were refit on the combined training and validation sets and used to generate forecasts for the 28-day holdout test window. Each model produced base forecasts $\hat{\boldsymbol{b}} \in \mathbb{R}^{m_L}$ for the bottom-level series. To ensure hierarchical coherence across all aggregation levels, these base forecasts were reconciled using the Minimum Trace (MinT) framework described in Chapter 2.

MinT reconciliation. Let $\boldsymbol{S} \in \mathbb{R}^{N \times m_L}$ denote the summing matrix linking bottom-level and aggregate series, and $\boldsymbol{W}_h = \text{Var}(\hat{\varepsilon}_{t+h})$ the base forecast error covariance matrix. The reconciled forecast at horizon h is given by:

$$\tilde{\boldsymbol{y}}_{t+h} = \boldsymbol{S}\big(\boldsymbol{S}^\top \boldsymbol{W}_h^{-1} \boldsymbol{S}\big)^{-1} \boldsymbol{S}^\top \boldsymbol{W}_h^{-1} \hat{\boldsymbol{y}}_{t+h}.$$

Following the variance structure assumed in [28], we used the structural approximation $\boldsymbol{W}_h = \boldsymbol{S}\boldsymbol{S}^\top$, which simplifies MinT to an Ordinary Least Squares (OLS) projection:

$$\tilde{\boldsymbol{y}}_{t+h} = \boldsymbol{S}\big(\boldsymbol{S}^\top \boldsymbol{S}\big)^{-1} \boldsymbol{S}^\top \hat{\boldsymbol{y}}_{t+h}.$$

This post-hoc reconciliation was implemented using `sktime.transformations.hierarchical.reconcile.Reconciler(`
It guarantees exact summation coherence while minimizing forecast variance within the assumed structural model. Alternative reconciliation strategies such as bottom-up and soft coherence regularization were also evaluated for comparison.

Forecast collection. For each model and forecast horizon $h = 1{:}28$, we collected:

- bottom-level forecasts $\hat{\boldsymbol{b}}_{t+h}$,
- reconciled forecasts $\tilde{\boldsymbol{y}}_{t+h}$ at all hierarchy levels,

This enabled level-wise comparison of both raw and reconciled forecasts in the evaluation stage.

4.2.8. Evaluate

Forecast accuracy was assessed following the M5 competition protocol, using both scale-normalized and sales-weighted metrics to ensure fair comparability across time series with different magnitudes. The metrics were the normalized Mean Absolute Error (nMAE) and the Weighted Root Mean Squared Scaled Error (WRMSSE). Weights were computed following the official M5 evaluation script.

Evaluation setup. All metrics were computed using the test window ($H = 28$) and aggregated across hierarchy levels using the summing matrix $\boldsymbol{S}$. Evaluation scripts were implemented using `sktime.performance_metrics.forecasting` and custom helper functions for WRMSSE. Model performance was reported both overall and per hierarchy level, providing insight into how well each model captured local and aggregate demand dynamics.

4.3. HGN Implementation Details

To implement the Hierarchical Graph Neural Network (HGN) model, we made several architectural choices guided by both theoretical considerations and empirical evaluations on validation data. This section details the specific implementation aspects of our model.

4.3.1. Architectural Specifications

The HGN architecture consists of 3 graph neural network layers with residual connections between consecutive layers to facilitate gradient flow during training. The propagation rule at each layer l is implemented as:

$$\boldsymbol{H}^{(l+1)} = \boldsymbol{H}^{(l)} + \sigma\left(\boldsymbol{W}_1^{(l)}\boldsymbol{H}^{(l)} + \boldsymbol{W}_2^{(l)}\hat{\boldsymbol{A}}\boldsymbol{H}^{(l)} + \boldsymbol{W}_3^{(l)}\boldsymbol{S}\hat{\boldsymbol{B}}^{(l)}\right) \tag{4.1}$$

where $\hat{\boldsymbol{A}} = \boldsymbol{D}_{\boldsymbol{A}}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}_{\boldsymbol{A}}^{-\frac{1}{2}}$ is the symmetrically normalized adjacency matrix for cross-series relationships with $\boldsymbol{D}_{\boldsymbol{A}}$ being the corresponding degree matrix, and $\hat{\boldsymbol{B}}^{(l)}$ is the projection of $\boldsymbol{H}^{(l)}$ to the bottom-level nodes. A visual representation of this architecture is provided in Appendix B, illustrating the information flow through multiple GNN layers and the interactions between hierarchical and cross-series components.

For the cross-series adjacency matrix $\boldsymbol{A}$, we implemented a hybrid approach:

1. For same-level adjacencies, we calculated Pearson correlation between time series over the training period and retained connections with absolute correlation $|\rho_{ij}| > 0.4$
2. For product hierarchies, we enforced connections between products in the same category
3. For geographical hierarchies, we connected stores within the same state

4.3.2. Network Dimensionality

The dimensionality choices for our network were:

- Input features: $d_0 = 42$, comprising 28 lagged values (representing 4 weeks of daily data), 7 day-of-week indicators, 4 week-of-month indicators, and 3 indicators for promotional periods
- Hidden dimensions: $d_1 = d_2 = d_3 = 64$ for all GNN layers. Keeping the dimensions equal ensures that the residual connections $H^{(l+1)} = H^{(l)} + \dots$ are valid
- Forecast dimension: the final MLP head maps the 64-dimensional embeddings of the bottom-level nodes to $h = 28$ forecasts (a 28-day horizon)

The weight matrices in the GNN layers therefore have dimensions

$$\boldsymbol{W}_1^{(l)}, \boldsymbol{W}_2^{(l)}, \boldsymbol{W}_3^{(l)} \in \mathbb{R}^{64\times 64} \quad \text{for } l \in \{0, 1, 2\}, \tag{4.2}$$

while the MLP head applies subsequent linear projections of dimensions $64 \to 64 \to 32 \to 28$.

4.3.3. Activation Functions and Regularization

We applied the following choices for activation functions and regularization:

- Activation function σ: We used ReLU (Rectivfied Linear Unit) activation, defined as $\text{ReLU}(x) = \max(0, x)$.
- Layer normalization was applied after each GNN layer to stabilize training
- Dropout with probability $p_{\text{drop}} = 0.2$ was applied to node features before each GNN layer to prevent overfitting
- L2 regularization with coefficient $\lambda_{\text{reg}} = 10^{-4}$ was added to the loss function to penalize large weights

For the final forecast generation, we implemented a bottom-level forecasting approach with an MLP head:

$$\hat{\boldsymbol{y}}_{i,1:h} = \text{MLP}(h_i^{(L)}) \tag{4.3}$$

The MLP consists of two hidden layers of dimensions 64 and 32 with ReLU activations.

4.3.4. Training Procedures

The model was trained using the following procedure:

- Optimization algorithm: Adam optimizer with initial learning rate $\eta = 10^{-4}$
- Batch size: 32 hierarchical substructures per batch
- Training epochs: 400, with early stopping based on validation performance with patience of 15 epochs
- Loss function: Mean Squared Error and a coherence penalty:

$$\mathcal{L} = \mathcal{L}_{\text{MSE}} + \lambda_{\text{coh}} \|\hat{\boldsymbol{y}} - \boldsymbol{S}\hat{\boldsymbol{b}}\|_2^2 \tag{4.4}$$

where $\lambda_{\text{coh}} = 0.1$ is the coherence penalty coefficient.

4.4. Results of benchmark

We conducted experiments with the M5 dataset to evaluate our demand sensing framework. The experiments were designed to assess two key aspects:

1. The performance of different forecasting models across various aggregation levels
2. The impact of including external drivers on forecast accuracy

4.4.1. Model Performance Comparison

Our first experiment focused on comparing the performance of various forecasting models on the M5 dataset without external drivers. Table 4.1 presents the normalized Mean Absolute Error (nMAE) for different models across various aggregation levels.

Table 4.1: Normalized Mean Absolute Error (nMAE) for Different Models without External Drivers

Model	L1	L2	L3	L4	L5	L12	AVG	Fitting Duration
Naïve	0.093	0.112	0.150	0.224	0.352	0.981	0.910	4s
1ML-ForEach								
AutoARIMA	0.055	0.066	0.101	0.199	0.272	0.862	0.738	15m 47s
Random Forest	0.053	0.079	0.123	0.196	0.354	0.985	0.806	189m 17s
LightGBM	0.057	0.090	0.150	0.213	0.305	0.884	0.786	155m 33s
LSTM	0.055	0.082	0.115	0.179	0.283	0.909	0.752	243m 47s
1ML-ForAll								
ARIMAX	0.061	0.083	0.123	0.227	0.262	0.887	0.814	12m 53s
Random Forest	0.068	0.092	0.164	0.267	0.312	0.943	0.863	27m 6s
LightGBM	0.071	0.084	0.129	0.222	0.233	0.861	0.773	19m 45s
LightGBM (opt.)	0.059	0.071	0.111	0.197	0.213	0.881	0.762	$\sim$7m
LSTFLinear	0.093	0.072	0.127	0.216	0.306	0.966	0.851	14m 34s
LSTFNLinear	0.084	0.073	0.131	0.209	0.284	0.865	0.842	16m 5s
TBATS	0.049	0.062	0.109	0.256	0.298	0.903	0.881	67m 29s
HGN (400 epochs)	0.105	0.120	0.187	0.212	0.229	0.877	0.784	105m 54s

Several key observations can be made from these results:

1. **Aggregation level impact**: All models show better performance at higher aggregation levels (L1, L2) compared to more granular levels (L12), confirming the theoretical principle that aggregation can reduce forecast error by smoothing out individual fluctuations.
2. **Model comparison**: Traditional methods like AutoARIMA perform competitively, particularly at higher aggregation levels. However, machine learning methods, especially LightGBM, show strong performance across all levels, with the optimized version achieving one of the best overall average performances.

3. **Computational considerations**: There is a clear trade-off between model complexity and computational requirements. While LSTM models achieve good accuracy, they require significantly longer fitting times compared to traditional methods like ARIMAX.
4. **Training strategy comparison**: The 1ML-ForEach approach (fitting separate models for each series) generally outperforms the 1ML-ForAll approach (fitting a single model to all series) in terms of accuracy, but at a much higher computational cost.
5. **HGN performance**: The Graph Neural Network model shows promising results, particularly at intermediate aggregation levels (L4, L5), suggesting that capturing relationships between time series can be beneficial.

4.4.2. HGN first tests

Our initial experiments with HGN for demand sensing showed promising results. By modeling the hierarchical structure of the M5 dataset as a graph, where nodes represent time series at different aggregation levels and edges represent aggregation relationships, we were able to capture both temporal patterns and cross-series dependencies.

The HGN model achieved competitive performance, particularly when incorporating external drivers, with an average nMAE of 0.731 compared to 0.784 without external drivers. This performance is comparable to other advanced methods like LSTM (0.703) and significantly better than the naive baseline (0.910).
Key observations from our HGN experiments include:

1. **Hierarchical relationships**: The HGN effectively captured hierarchical relationships, performing particularly well at intermediate aggregation levels (L4, L5) where both parent-child and sibling relationships provide valuable information.
2. **Computational efficiency**: While HGN training requires significant computational resources (128m 18s fitting duration), the prediction is relatively fast (35s), making it suitable for operational settings where models are retrained periodically but predictions are needed frequently.
3. **External driver integration**: The HGN architecture naturally accommodates external drivers as node features, allowing it to leverage both structural relationships and exogenous variables.
4. **Further potential**: Our initial HGN implementation shows promise, but further improvements may be possible through architectural refinements, more sophisticated graph construction, and better hyperparameter tuning.

These observations highlight both the promise and the open challenges of applying HGNs to hierarchical retail forecasting, motivating the refinements and systematic evaluations presented in the following sections.

4.4.3. Impact of External Drivers

Our second experiment evaluated the impact of incorporating external drivers into the forecasting models. Table 4.2 presents the normalized Mean Absolute Error (nMAE) for different models with external drivers included.

Comparing Tables 4.1 and 4.2 reveals the following insights:

1. **Improved accuracy with external drivers**: Across almost all models and aggregation levels, incorporating external drivers leads to significant improvements in forecast accuracy. For example, the average nMAE for AutoARIMA improves from 0.738 to 0.722 when including external variables.
2. **Varying impact across models**: The improvement from including external drivers varies across models. Machine learning models like Random Forest and LightGBM show substantial improvements, particularly at lower aggregation levels, highlighting their ability to effectively utilize complex relationships between external factors and demand.
3. **Optimal model selection**: When considering both accuracy and computational efficiency, the optimized LightGBM model with external drivers achieves the best overall performance with an average nMAE of 0.663.

Table 4.2: Normalized Mean Absolute Error (nMAE) for Different Models with External Drivers

Model	L1	L2	L3	L4	L5	L12	AVG	Fitting Duration
Naïve	0.093	0.112	0.150	0.224	0.352	0.981	0.910	4s
1ML-ForEach								
AutoARIMAX	0.044	0.049	0.084	0.123	0.202	0.844	0.722	49m 46s
Random Forest	0.043	0.076	0.101	0.161	0.242	0.832	0.767	266m 4s
LightGBM	0.049	0.074	0.105	0.150	0.249	0.792	0.732	208m 54s
LSTM	0.039	0.060	0.091	0.146	0.223	0.805	0.703	307m 53s
1ML-ForAll								
ARIMAX	0.042	0.065	0.105	0.209	0.284	0.841	0.735	17m 23s
Random Forest	0.059	0.055	0.092	0.154	0.228	0.807	0.744	38m 30s
LightGBM	0.060	0.053	0.099	0.158	0.239	0.796	0.765	25m 53s
LightGBM (opt.)	0.057	0.046	0.087	0.168	0.246	0.715	0.663	$\sim$10m
LSTFLinear	0.086	0.066	0.104	0.153	0.304	0.916	0.728	19m 52s
LSTFNLinear	0.077	0.065	0.103	0.151	0.287	0.847	0.683	20m 23s
TBATS	0.028	0.053	0.083	0.157	0.237	0.755	0.698	94m 18s
HGN (400 epochs)	0.046	0.052	0.115	0.194	0.295	0.790	0.731	128m 18s

4. **HGN improvements**: The Graph Neural Network model shows improvements when incorporating external drivers, with the average nMAE decreasing from 0.784 to 0.731. This suggests that HGNs can effectively leverage both temporal dependencies and external factors.

4.4.4. Weighted Root Mean Squared Scaled Error Results

In addition to nMAE, we evaluated model performance using the Weighted Root Mean Squared Scaled Error (WRMSSE), the primary evaluation metric used in the M5 competition. Table 4.3 presents these results for selected models. WRMSSE weights and scales follow the M5 definition and are computed on the evaluation subset used in our experiments.

Table 4.3: Weighted Root Mean Squared Scaled Error (WRMSSE) for Selected Models with External Drivers

Model	L1	L2	L3	L4	L5	L12	AVG
Naïve	1.564	1.417	1.529	1.472	1.525	1.505	1.502
1ML-ForEach							
AutoARIMAX	0.645	0.958	1.191	1.416	1.474	1.460	1.191
Random Forest	0.641	0.774	0.999	1.143	1.104	1.101	0.960
LightGBM	0.713	0.712	1.037	1.127	1.146	1.113	0.978
1ML-ForAll							
ARIMAX	0.784	1.035	0.964	1.354	1.023	1.264	1.071
Random Forest	0.840	0.703	0.898	0.986	0.989	1.103	0.920
LightGBM (opt.)	0.939	1.207	1.059	1.028	0.899	0.619	0.822

The WRMSSE results generally align with the nMAE findings, confirming the superior performance of machine learning models when external drivers are incorporated.

4.4.5. Model Prevalence Analysis

To further understand model performance across different time series, we analyzed how often each model was favored (i.e., achieved the lowest error) across the dataset. Figure 4.2 illustrates the distribution of best-performing models for individual SKUs.

This analysis reveals that:

1. In the base setting (without external drivers), traditional methods like AutoARIMA are frequently the best performers, particularly for series with regular patterns.

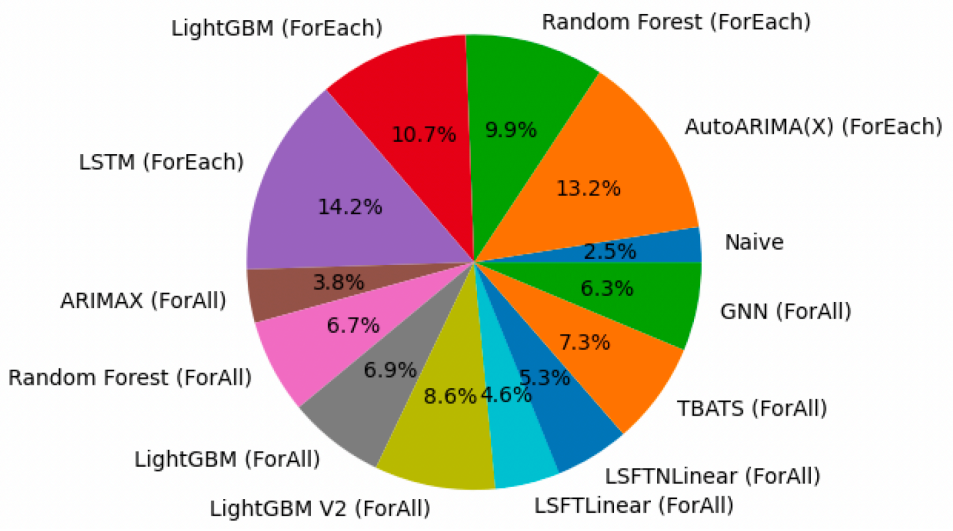


Figure 4.2: Distribution of Best-Performing Models Across SKUs

2. When external drivers are incorporated, machine learning methods like LightGBM and Random Forest become more prevalent as the best performers, highlighting their ability to effectively utilize additional information.
3. No single model dominates across all time series, underscoring the value of a diverse forecasting toolkit that can be adapted to different data characteristics.

4.4.6. Correlation Analysis

The performance improvements observed when incorporating external drivers can be better understood by examining the correlations between sales and these external factors. Figures 4.3 and 4.4 shows a heatmap of Pearson correlation coefficients between sales and various external drivers.

Key observations from the correlation analysis include:

1. Positive correlation between sales and product price history, potentially indicating higher-priced items selling in larger volumes during certain periods.
2. Negative correlation between sales and unemployment rate, reflecting the relationship between economic conditions and consumer spending.
3. Negative correlation between sales and certain weather conditions, suggesting weather influences on shopping patterns.
4. Complex inter-SKU correlations, with certain zones showing either positive or negative correlations, indicating potential complementary or substitute relationships between products.

4.4.7. Forecasting Horizon Analysis

We also evaluated how forecast accuracy changes with the forecasting horizon. Figure 4.5 displays the distribution of nMAE for various forecasting periods for both the 1ML-ForEach and 1ML-ForAll approaches.
The analysis showed that:

1. Mean nMAE shifts rightward with longer forecasting periods, indicating increased volatility or variability in the forecasts.
2. The distribution of errors becomes wider at longer horizons, consistent with the accumulation of uncertainty over time.
3. The performance degradation with increasing horizon is less severe for models incorporating external drivers, suggesting that these variables provide valuable information for longer-term forecasting.

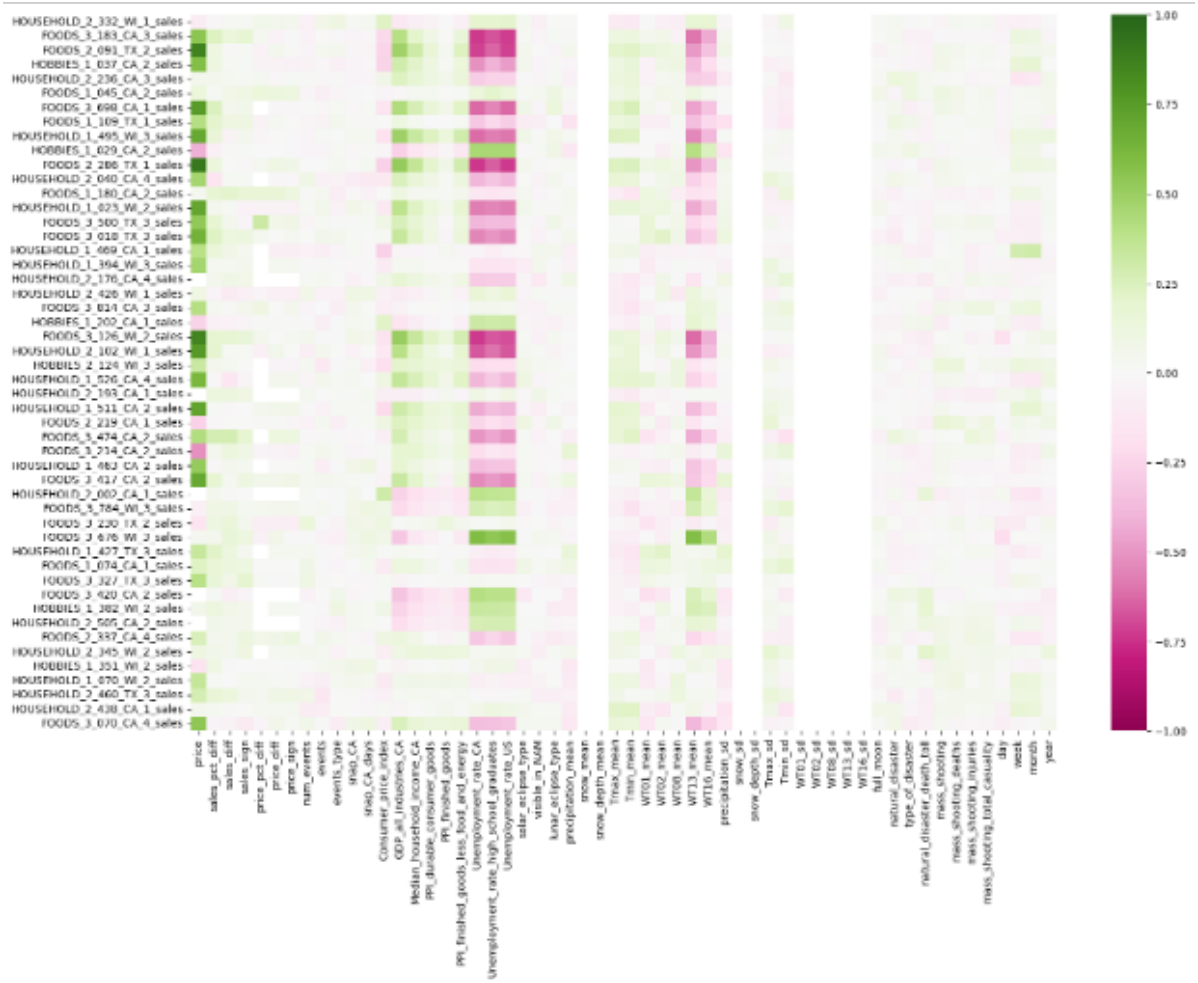


Figure 4.3: Correlation Between Sales and External Drivers (1ML-ForEach)

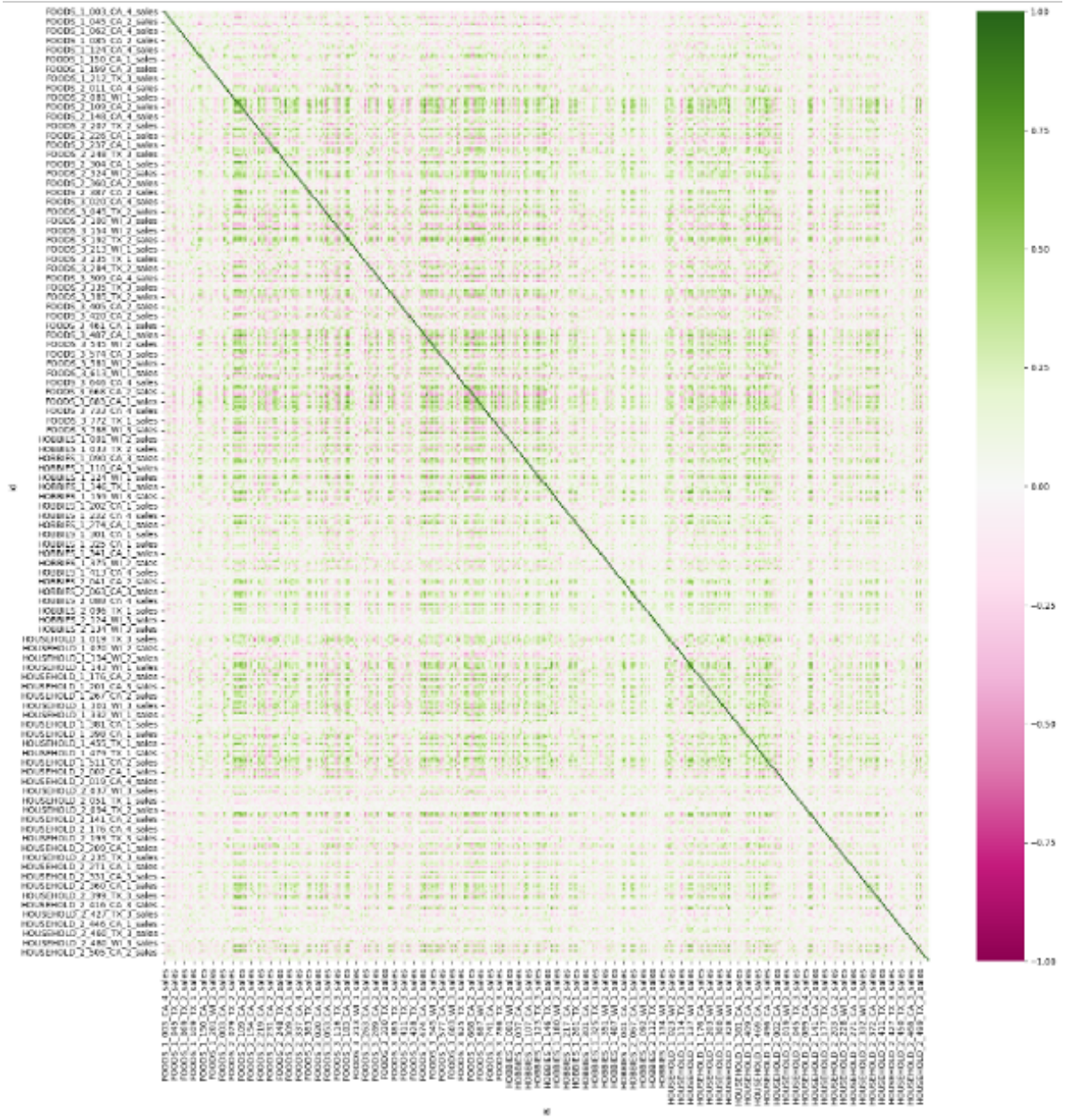


Figure 4.4: Correlation Between Sales and External Drivers (1ML-ForAll).

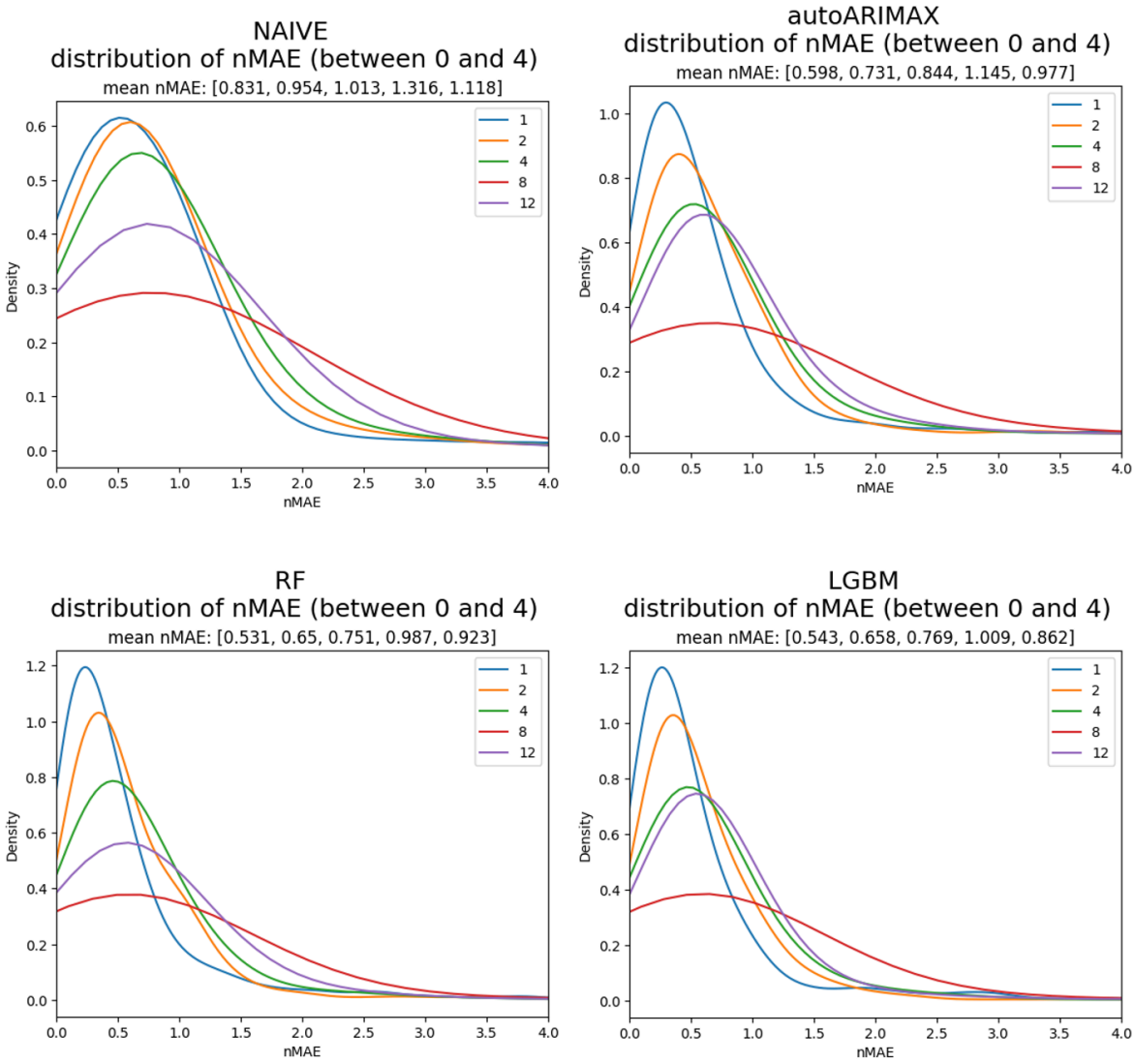


Figure 4.5: nMAE Distribution for Various Forecasting Horizons

4.5. Discussion and Future Work

The experiments conducted on the M5 dataset provide valuable insights into the performance of different forecasting approaches and the impact of external drivers on forecast accuracy. However, there are several limitations to our current implementation and opportunities for future research that are worth discussing before proceeding to the final conclusions in Chapter 5.

4.5.1. Limitations of Current Implementation

Despite the promising results obtained with our demand sensing framework, several limitations should be acknowledged:

1. **Computational Scalability**: While our experiments used a substantial subset of the M5 dataset (1,000 SKUs), scaling to the full dataset (30,490 SKUs) requires significant computational resources, particularly for models like LSTM and HGNs. This poses challenges for real-time or near-real-time forecasting in large-scale retail operations.
2. **Feature Engineering**: Our current implementation relies on manually engineered features for incorporating external drivers. This approach may not capture all relevant patterns and interactions, especially in complex retail environments with numerous influencing factors.
3. **Limited External Data**: Although we incorporated several external drivers (weather, economic indicators, events), the lack of inventory information and detailed product attributes in the M5 dataset limits our ability to capture certain demand drivers that might be critical in real-world settings.
4. **HGN Architecture Exploration**: Our initial HGN implementation showed promising results, but we explored only a limited set of graph architectures and attention mechanisms. More sophisticated graph construction and neural network architectures might yield further improvements.

4.5.2. Directions for Future Work

Based on the insights gained from our experiments and the limitations identified, several promising directions for future research emerge:

1. **Advanced HGN Architectures**: Exploring more sophisticated HGN architectures that can better capture temporal dynamics and hierarchical relationships is a natural extension of our work. Techniques such as temporal graph attention networks or hierarchical graph embeddings might offer improved performance.
2. **End-to-End Learning**: Developing end-to-end learning approaches that jointly optimize feature extraction, forecasting, and reconciliation could reduce the need for manual feature engineering while potentially improving overall accuracy.
3. **Automated Model Selection**: Our results indicate that no single model dominates across all time series and aggregation levels. Developing automated model selection or ensemble techniques that adapt to different data characteristics could further improve forecast accuracy.
4. **Explainable AI Integration**: While some models like Random Forest and LightGBM offer feature importance measures, enhancing the interpretability of more complex models, particularly HGNs, would make them more accessible to practitioners in business settings.
5. **Uncertainty Quantification**: Extending our framework to provide probabilistic forecasts with reliable uncertainty estimates would enhance decision-making in supply chain contexts, where understanding risk is crucial.

Building on these experimental results and insights, Chapter 5 will provide conclusions regarding the theoretical and practical contributions of our demand sensing framework, along with broader implications for the field of demand forecasting in supply chain management.

5 Conclusion

This chapter arranges the findings of the thesis and reflects on their implications for both research and practice. It revisits the guiding research questions, summarizes the contributions of each chapter, and outlines directions for future work.

5.1. Recap of Research Objectives

The thesis addressed two guiding research questions:

1. How can hierarchical demand forecasting be formulated as a graph learning problem while ensuring coherent forecasts across all levels?
2. To what extent can graph neural networks improve forecast accuracy compared to classical and machine learning methods, especially when incorporating external drivers?

To investigate these questions, the thesis developed a forecasting framework grounded in supervised learning and hierarchical coherence, extended it with graph-based estimators, and evaluated the resulting models on the M5 dataset. We address both questions below.

Hierarchical forecasting can be cast as a *graph-based multi-product demand process*, where nodes represent series, edges encode parent–child and lateral relations, and $g_i(\cdot, G)$ is learned via message passing. Concretely, we (i) mapped the hierarchy to a graph $G = (V, E)$ with L-lag neighborhoods, (ii) implemented a Hierarchical Graph Network (HGN) that combines GNN layers with a bottom-level MLP head, and (iii) incorporated three coherence mechanisms: bottom-up construction, soft coherence regularization, and post-hoc MinT reconciliation. In practice, exact coherence was only guaranteed post hoc, while training-time regularization nudged the network toward respecting the hierarchy.

On the M5 dataset, the optimized global LightGBM with external drivers achieved the best overall average nMAE (0.663), outperforming both classical baselines and our HGN. The HGN was nonetheless competitive, improving from 0.784 (no drivers) to 0.731 (with drivers), and performed particularly well at intermediate aggregation levels where both parent–child and sibling information matter most. External drivers improved all methods, with the largest gains at granular levels for ML-based models. Local (1ML-ForEach) models often edged out global ones in accuracy, but at substantially higher computational cost.

5.2. Key Findings and Contributions

The contributions of the thesis can be organized according to its main components.

Foundational Framework (Chapter 2)

We introduced the *multi-product demand process* as a mathematical formalization of supply chain forecasting. The forecast operator provided a general way to express how models map historical information and external covariates to predictions. Classical models (e.g., ARIMA, ETS), machine learning

estimators (e.g., LightGBM, LSTMs).

Hierarchical forecasting was formalized as a problem of ensuring coherence across aggregation levels, with detailed proofs and reconciliation methods such as MinT positioned as projections within this framework.

Graph-Based Framework (Chapter 3)

We extended the operator-based forecasting framework to graphs, introducing a graph demand process that incorporates relational structure into the modeling setup. Chapter 3 formalized how nodes (time series) and edges (hierarchical or relational links) define local information neighborhoods, and how Graph Neural Networks (GCNs, GATs) can parameterize the resulting demand processes. Building on this, we outlined a practical pipeline for hierarchical GNN forecasting, including graph construction, feature design, message passing, and reconciliation strategies. This provides a principled foundation for applying graph-based learning to hierarchical demand forecasting.

Empirical Evaluation (Chapter 4)

Using the M5 dataset, we benchmarked classical methods, machine learning baselines, and the proposed HGN under both local (1ML-ForEach) and global (1ML-ForAll) training strategies, with and without external drivers. The main findings are:

- *Relative performance.* Machine learning models—particularly LightGBM—tend to outperform classical time-series models when many related series and external drivers are available, though classical models (e.g., AutoARIMA/TBATS) remain competitive at higher aggregation levels. In our setup, the optimized global LightGBM with drivers achieved the best overall average nMAE (0.663).
- *Value of external drivers.* Adding calendar, price, event, and macro features improved accuracy across almost all models and levels (e.g., HGN avg. nMAE improved from 0.784 to 0.731; AutoARIMA $\rightarrow$ AutoARIMAX from 0.738 to 0.722). Gains were generally larger for ML models at granular levels.
- *Local vs. global training.* Local (1ML-ForEach) often achieved stronger accuracy but at substantially higher compute cost; global (1ML-ForAll) offered a better accuracy–efficiency trade-off when paired with a strong learner (e.g., optimized LightGBM).
- *Horizon effects.* Error distributions shifted right and widened with longer horizons, as expected. Models with external drivers degraded more slowly, indicating a stabilizing effect at medium horizons.
- *HGN performance and coherence.* The HGN showed promising accuracy, especially at intermediate aggregation levels where parent–child and sibling relations are informative. In line with our design, we encouraged coherence during training (regularization) and enforced exact coherence post hoc via MinT reconciliation.

5.3. Implications

Theoretical Implications

The operator-based framing unifies classical forecasting, machine learning, and graph-based models under a single formalism. This perspective clarifies how design choices on information scope and parameter sharing position different estimators, enabling more rigorous comparisons. Furthermore, the exploration of HGNs shows how graph learning can be extended to hierarchical forecasting, though challenges remain in ensuring exact coherence and scaling to large product assortments.

Practical Implications

For practitioners, three key insights stand out. First, incorporating external drivers yields systematic improvements and requires robust data infrastructure. Second, the choice between per-series and global models involves trade-offs between flexibility and data efficiency; hybrid approaches such as clustering offer a middle ground. Third, reconciliation remains essential in operational settings, even when models attempt to embed hierarchy directly, to guarantee consistent forecasts across aggregation levels.

5.4. Concluding Remarks

This thesis developed a demand sensing framework that bridges classical time series methods, modern machine learning, and graph-based approaches. Through theoretical formalization and empirical testing, it demonstrated both the opportunities and the limitations of applying graph neural networks to hierarchical forecasting. While HGNs show promise, especially in capturing intermediate-level dependencies, their practical deployment requires further research into scalable graph construction, probabilistic coherence, and end-to-end training methods.

Looking ahead, future work should explore richer representations of external drivers, probabilistic extensions of HGNs, and hybrid pipelines that combine the interpretability of classical models with the flexibility of deep graph architectures. As supply chains continue to face volatility and complexity, the integration of hierarchical structure, external signals, and advanced machine learning offers a promising path toward more accurate, coherent, and responsive demand forecasting.

References

[1] Ilan Alon, Min Qi, and Robert J Sadowski. “Forecasting aggregate retail sales:: a comparison of artificial neural networks and traditional methods”. In: *Journal of retailing and consumer services* 8.3 (2001), pp. 147–156.

[2] George Athanasopoulos and Nikolaos Kourentzes. “On the evaluation of hierarchical forecasts”. In: *International Journal of Forecasting* 39.4 (2023), pp. 1502–1511.

[3] Kasun Bandara, Christoph Bergmeir, and Slawek Smyl. “Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach”. In: *Expert systems with applications* 140 (2020), p. 112896.

[4] Gerard P Cachon and Marshall Fisher. “Supply chain inventory management and the value of shared information”. In: *Management science* 46.8 (2000), pp. 1032–1048.

[5] Alysha M De Livera, Rob J Hyndman, and Ralph D Snyder. “Forecasting time series with complex seasonal patterns using exponential smoothing”. In: *Journal of the American statistical association* 106.496 (2011), pp. 1513–1527.

[6] Robert Fildes, Paul Goodwin, and Dilek Önkal. “Use and misuse of information in supply chain forecasting of promotion effects”. In: *International Journal of Forecasting* 35.1 (2019), pp. 144–156.

[7] Gene Fliedner. “An investigation of aggregate variable time series forecast strategies with specific subaggregate time series statistical correlation”. In: *Computers & operations research* 26.10-11 (1999), pp. 1133–1149.

[8] Charles W Gross and Jeffrey E Sohl. “Disaggregation methods to expedite product line forecasting”. In: *Journal of forecasting* 9.3 (1990), pp. 233–254.

[9] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. “Wavelets on graphs via spectral graph theory”. In: *Applied and Computational Harmonic Analysis* 30.2 (2011), pp. 129–150.

[10] RJ Hyndman. *Forecasting: principles and practice*. OTexts, 2018.

[11] Rob J Hyndman et al. “Optimal combination forecasts for hierarchical time series”. In: *Computational statistics & data analysis* 55.9 (2011), pp. 2579–2589.

[12] Dmitry Ivanov, Alexandre Dolgui, and Boris Sokolov. “The impact of digital technology and Industry 4.0 on the ripple effect and supply chain risk analytics”. In: *International journal of production research* 57.3 (2019), pp. 829–846.

[13] Jianhua Ji et al. “The Impact of Demand Correlation on Bullwhip Effect in a Two-stage Supply Chain with Two Retailers”. In: *Proceedings of the International Conference on Operations Research and Enterprise Systems - Volume 1: ICORES,* INSTICC. SciTePress, 2015, pp. 304–313. ISBN: 978-989-758-075-8. DOI: 10.5220/0005233003040313.

[14] Thomas N Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks”. In: *arXiv preprint arXiv:1609.02907* (2016).

[15] Yaguang Li et al. “Diffusion convolutional recurrent neural network: Data-driven traffic forecasting”. In: *arXiv preprint arXiv:1707.01926* (2017).

[16] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. “The M5 competition: Background, organization, and implementation”. In: *International Journal of Forecasting* 38.4 (2022), pp. 1325–1336.

[17] Spyros Makridakis et al. *The M5 Competition: Competitors’ Guide*. Tech. rep. Makridakis Open Forecasting Center (MOFC), 2020. URL: https://mofc.unic.ac.cy/m5-competition/.

[18] Pablo Montero-Manso and Rob J Hyndman. “Principles and algorithms for forecasting groups of time series: Locality and globality”. In: *International Journal of Forecasting* 37.4 (2021), pp. 1632–1653.

[19] Konstantinos Nikolopoulos et al. “Forecasting and planning during a pandemic: COVID-19 growth rates, supply chain disruptions, and governmental decisions”. In: *European journal of operational research* 290.1 (2021), pp. 99–115.

[20] Boris N Oreshkin et al. “N-BEATS: Neural basis expansion analysis for interpretable time series forecasting”. In: *arXiv preprint arXiv:1905.10437* (2019).

[21] Nikolay Osadchiy, Vishal Gaur, and Sridhar Seshadri. “Sales forecasting with financial indicators and experts’ input”. In: *Production and Operations Management* 22.5 (2013), pp. 1056–1076.

[22] Fotios Petropoulos and Nikolaos Kourentzes. “Forecast combinations for intermittent demand”. In: *Journal of the Operational Research Society* 66.6 (2015), pp. 914–924.

[23] Fotios Petropoulos et al. “Forecasting: theory and practice”. In: *International Journal of Forecasting* 38.3 (2022), pp. 705–871.

[24] Syama Sundar Rangapuram et al. “Deep state space models for time series forecasting”. In: *Advances in neural information processing systems* 31 (2018).

[25] Syama Sundar Rangapuram et al. “End-to-end learning of coherent probabilistic forecasts for hierarchical time series”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8832–8843.

[26] Aris A Syntetos, John E Boylan, and JD Croston. “On the categorization of demand patterns”. In: *Journal of the operational research society* 56 (2005), pp. 495–503.

[27] Aris A Syntetos et al. “Supply chain forecasting: Theory, practice, their gap and the future”. In: *European Journal of Operational Research* 252.1 (2016), pp. 1–26.

[28] Shanika L Wickramasuriya, George Athanasopoulos, and Rob J Hyndman. “Optimal forecast reconciliation for hierarchical and grouped time series through trace minimization”. In: *Journal of the American Statistical Association* 114.526 (2019), pp. 804–819.

[29] Zonghan Wu et al. “Connecting the dots: Multivariate time series forecasting with graph neural networks”. In: *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 2020, pp. 753–763.

[30] Zonghan Wu et al. “Graph wavenet for deep spatial-temporal graph modeling”. In: *arXiv preprint arXiv:1906.00121* (2019).

[31] Bing Yu, Haoteng Yin, and Zhanxing Zhu. “Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting”. In: *arXiv preprint arXiv:1709.04875* (2017).

A

HGN Architecture

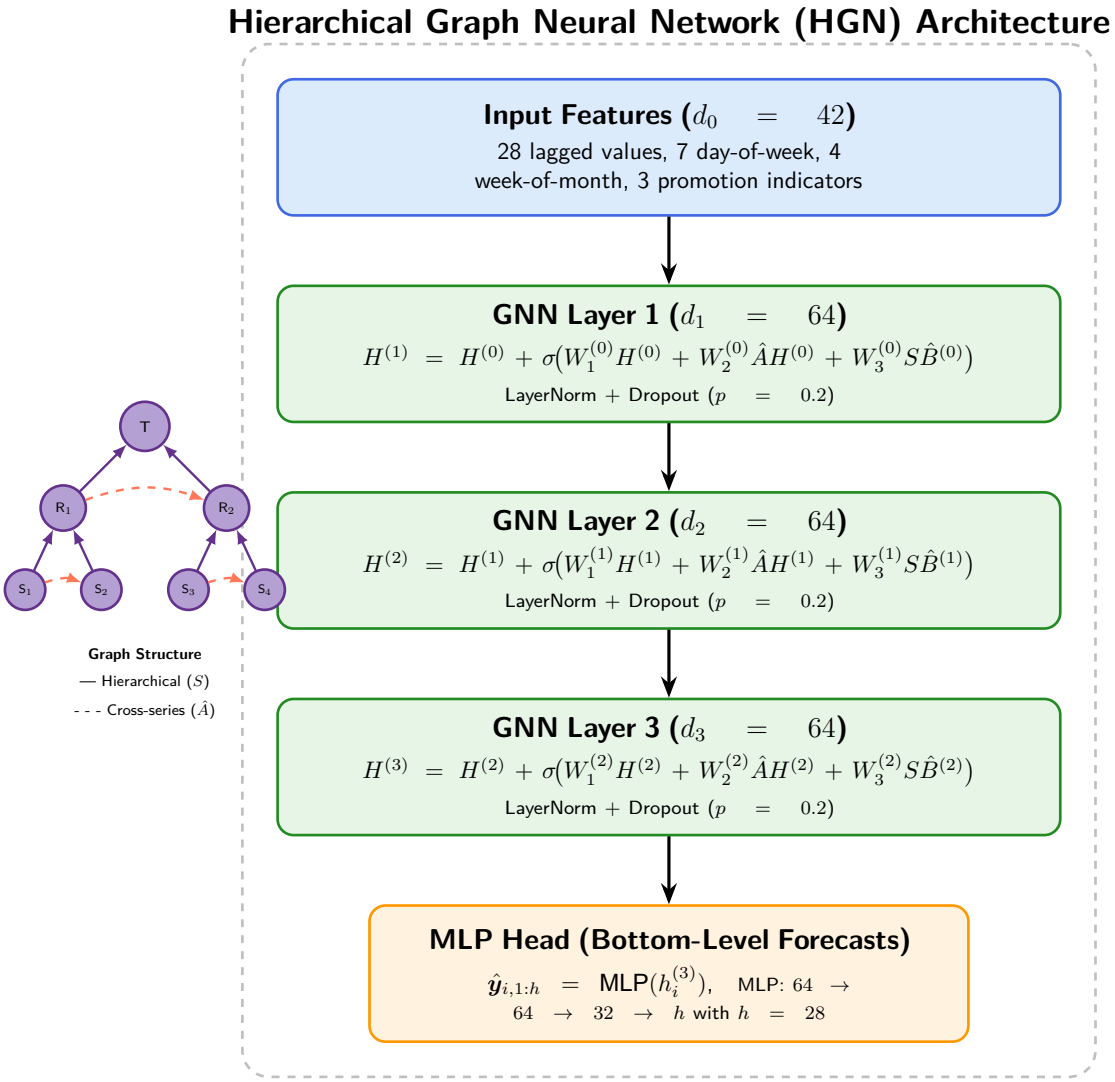


Figure A.1: HGN architecture consistent with Chapter 4: input features $d_0 = 42$; three residual GNN layers with $d_1 = d_2 = d_3 = 64$; bottom-level MLP head maps 64-dimensional embeddings to $h = 28$ forecasts. Coherence is enforced via bottom-up, regularization, or post-hoc reconciliation (MinT).