Learning Production Functions for Supply Chains with Graph Neural Networks

Serina Chang[1], Zhiyin Lin[1], Benjamin Yan[1], Swapnil Bembde[2], Qi Xiu[2], Chi Heem Wong[1,2], Yu Qin[2,3], Frank Kloster[2], Xi Luo[2], Raj Palleti[1,2], Jure Leskovec[1]

[1]Stanford University, Department of Computer Science
[2]Hitachi America, Ltd.
[3]Tulane University

Abstract

The global economy relies on the flow of goods over supply chain networks, with nodes as firms and edges as transactions between firms. While we may observe these external transactions, they are governed by unseen *production functions*, which determine how firms internally transform the input products they receive into output products that they sell. In this setting, it can be extremely valuable to infer these production functions, to improve supply chain visibility and to forecast future transactions more accurately. However, existing graph neural networks (GNNs) cannot capture these hidden relationships between nodes' inputs and outputs. Here, we introduce a new class of models for this setting by combining temporal GNNs with a novel inventory module, which learns production functions via attention weights and a special loss function. We evaluate our models extensively on real supply chains data and data generated from our new open-source simulator, `SupplySim`. Our models successfully infer production functions, outperforming the strongest baseline by 6%–50% (across datasets), and forecast future transactions, outperforming the strongest baseline by 11%–62%.

Code — https://github.com/snap-stanford/supply-chains
Extended Version — https://arxiv.org/pdf/2407.18772

1 Introduction

Supply chains form the backbone of the global economy and disruptions can have enormous consequences, costing trillions of dollars (Baumgartner, Malik, and Padhi 2020) and risking national security (The White House 2021). Thus, modeling supply chains and how they evolve, especially under shocks, is essential. Prior models of supply chains have been mainly mechanistic and struggle to fit even highly aggregated measures, such as country-level production (Inoue and Todo 2019). As a result, recent literature has called for integration of machine learning (ML) into supply chain modeling (Baryannis, Dani, and Antoniou 2019; Brintrup et al. 2020; Younis, Sundarakani, and Alsharairi 2022).

Graph neural networks (GNNs) serve as a particularly promising ML methodology, as supply chains are naturally represented as graphs, with nodes as firms and edges as transactions between firms. A few works have explored static GNNs for supply chains (Aziz et al. 2021; Kosasih and Brintrup 2021; Wasi, Islam, and Akib 2024), but these static models miss crucial dynamic aspects of supply chains, such as the propagation of shocks. Thus, there is a need to develop temporal GNNs that can capture the unique dynamics and mechanisms of supply chain graphs.

One key feature of these graphs is that they are governed by underlying *production functions*: firms receive input products (e.g., wheels) from other firms, internally transform those inputs into outputs (e.g., cars) via production functions, then sell those output products to other firms or consumers (Carvalho and Tahbaz-Salehi 2019). These production functions dictate which firms are connected to each other, as well as the timing of transactions across the network. For example, if a firm experiences a shortage in an input, its production of outputs that rely on that input will be disrupted, but it can continue to produce other outputs for some time. However, existing temporal GNNs (Huang et al. 2023) are not designed to learn these hidden production functions or to incorporate them into link prediction.

The present work. Here, we are among the first to develop temporal GNNs for supply chains, demonstrating their ability to learn rich, dynamic representations of firms and products. Furthermore, by introducing supply chains to temporal GNNs, we identify a challenging setting unexplored by prior models: temporal graphs governed by production functions, which we term *temporal production graphs* (TPGs). Beyond supply chains, TPGs also appear in biological domains, such as enzymes producing metabolites within metabolic pathways, or in organizations, where teams rely on inputs from other teams to produce their outputs. TPGs introduce new challenges compared to other temporal graphs, such as the need to infer production functions. Furthermore, even standard tasks, like link prediction, may be complicated under TPGs. For example, while standard temporal GNNs might capture disruptions generally propagating across connected firms, they will miss the specific connections between each firm's inputs and outputs and, under a shortage of inputs, not be able to precisely predict which outputs will be affected.

Since existing GNNs cannot handle TPGs, we introduce a *new class of GNNs* designed for TPGs, focusing on two objectives: (1) learning the graph's production functions, (2) predicting its future edges. Prior temporal GNNs have

focused primarily on the second objective, but are not designed for the first. Our models support both objectives, by combining temporal GNNs with a novel inventory module, which learns production functions by explicitly representing each firm's inventory and updating it based on attention weights that map external supply to internal consumption. Our module can be combined with *any* GNN; to demonstrate this, here we combine it with two popular models, Temporal Graph Network (Rossi et al. 2020) and GraphMixer (Cong et al. 2023), which we also extend in important ways.

We demonstrate the utility of our models on real and synthetic supply chains data. We have rare access to real *transaction-level* supply chains data, which allows us to evaluate our GNNs' abilities to predict individual transactions. However, since we cannot release the proprietary data, we also build a new open-source simulator, `SupplySim`, which generates realistic supply chain data that matches the real data on key characteristics. Our simulator enables us to share data, test models under controlled settings (e.g., supply shocks, missing data), and encourage future research in this area. In summary, our contributions are:

1. **Problem**: a new graph ML problem setting, *temporal production graphs* (TPGs), where each node's edges are related via unobserved production functions,
2. **Models**: a new class of models for TPGs, which combine temporal GNNs with a novel inventory module to jointly learn production functions and forecast future edges,
3. **Data**: an open-source simulator, `SupplySim`, which generates realistic supply chain data and enables model testing under controlled and varied settings,
4. **Results**: experiments on real and synthetic data, showing that our models effectively learn production functions (outperforming baselines by 6-50%) and forecast future edges (outperforming baselines by 11-62%).

Our work both contributes to the supply chains domain, by developing temporal GNNs for supply chains, and advances graph ML, by introducing a new type of real-world graph and developing new methods to handle them.

Social impact. Our work is a collaboration with Hitachi America, Ltd., a multinational conglomerate that produces a wide range of products, each involving a complex global supply chain. Our project was initiated by their need for ML solutions to supply chain problems, and together, we identified two ML objectives grounded in real business needs. Our first objective of learning production functions aims to improve supply chain visibility, so that firms can better understand the entire production process through which their products are produced (instead of only their immediate suppliers) and find more efficient solutions (Aigner and Chu 1968; Coelli et al. 2005). Our second objective of forecasting future transactions supports critical industry tasks, such as demand forecasting, early detection of risks, and inventory optimization. Additionally, our collaboration provides access to transactions-level data, which enables us to rigorously evaluate our ML models and build a realistic simulator so that, despite the data sharing constraints of this domain, we can still encourage future ML research on supply chains.

2 Related Work

Many real-world systems can be represented as temporal graphs, such as transportation systems (Jiang and Luo 2022), human mobility (Chang et al. 2023), and biological networks (Prill, Iglesias, and Levchenko 2005). While most GNNs are designed for static graphs, there has been growing interest recently in developing GNNs for temporal graphs (Huang et al. 2023; Skarding, Gabrys, and Musial 2021; Longa et al. 2023). However, to the best of our knowledge, GNNs have not yet been designed for temporal production graphs (TPGs), as described in this work. Furthermore, only a handful of works have explored GNNs for supply chains, all in static settings, such as to predict hidden links between firms (Aziz et al. 2021; Kosasih and Brintrup 2021), classify a firm's industry (Wu, Wang, and Olson 2023), recommend suppliers (Tu et al. 2024), and predict product relations (e.g., same product group) (Wasi, Islam, and Akib 2024).

Within the supply chains domain, our work builds on prior literature that represents supply chains as networks (Fujiwara and Aoyama 2010) and models the propagation of shocks over these networks (Acemoglu et al. 2012; Zhao, Zuo, and Blackhurst 2019; Li et al. 2020; Carvalho et al. 2021). Our work is unique in two key ways: first, much of the prior work relies on theoretical models and synthetic networks, and, even among empirical studies, the data used is typically industry-level or, at best, firm-level with static relations between firms (Carvalho and Tahbaz-Salehi 2019). In contrast, we have access to *transaction*-level data, which reveals essential time-varying information. Second, integration of ML into supply chains has been limited, with calls for further exploration (Baryannis, Dani, and Antoniou 2019; Brintrup et al. 2020; Younis, Sundarakani, and Alsharairi 2022). Most prior models for modeling supply chains are mechanistic (Hallegatte 2008; Guan et al. 2020; Inoue and Todo 2020; Li et al. 2021) and limited in their ability to even fit aggregate counts (Inoue and Todo 2019). In contrast, by combining our inventory module with GNNs, we can accurately forecast individual transactions, while maintaining the interpretability of the inventory module.

This theme of combining deep learning with domain-specific principles also appears in physics simulation (Wang, Walters, and Yu 2021) and epidemiological forecasting (Liu et al. 2024). Our work also has connections to temporal causal discovery (Nauta, Bucur, and Seifert 2019; Löwe et al. 2022; Assaad, Devijver, and Gaussier 2022) and inferring networks from node marginals (Kumar et al. 2015; Maystre and Grossglauser 2017; Chang et al. 2024).

3 Learning on Temporal Production Graphs

3.1 Problem Definition

We define a new graph ML setting called *temporal production graphs* (TPGs), illustrated in Figure 1a. TPGs are directed graphs with time-varying edges and potentially a time-varying set of nodes. What differentiates TPGs from other temporal graphs is that, in a TPG, each node's in-edges represent inputs to some internal production function, which are transformed into outputs, represented by the node's out-edges. In this work, we focus on supply chain networks, as

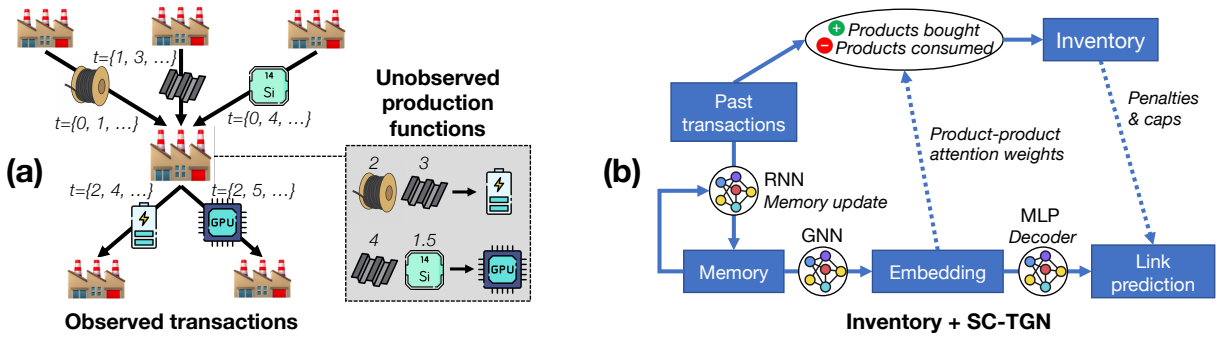

Figure 1: **(a)** Illustration of our problem setting: we observe time-varying transactions between firms and do not observe production functions within firms. Our goals are to learn the production functions and predict future transactions. **(b)** Example of our model architecture, combining our inventory module with our extended version of TGN, SC-TGN.

a canonical example of TPGs. We are given a set of transactions $\mathcal{T}$ between firms where, for each transaction, we know its timestamp, supplier firm, buyer firm, product sold, and amount sold. We represent this data as a heterogeneous temporal graph, $G_{\text{txns}} = \{\mathcal{N}, \mathcal{E}\}$, where the nodes $\mathcal{N}$ consist of n firm nodes and m product nodes and the edges are $\mathcal{E} := \{e(s, b, p, t)\}$, where $e(s, b, p, t)$ is a *hyperedge* between supplier firm s, buyer firm b, and product p, representing a transaction between them at time t.

In this setting, production functions define how firms internally transform the products that they buy into products that they supply. Specifically, the function $\mathcal{F}_p : \mathbb{R}_+ \to \mathbb{R}_+^m$ for product p defines how much of each product is necessary to make k amount of p (e.g., one car requires four wheels). The set of production functions define a *production graph*, G_{prod}, which is a directed acyclic graph where there is an edge from products p_1 to p_2 if p_1 is required to make p_2.[1] Given the set of transactions $\mathcal{T}$, and the resulting temporal graph G_{txns}, our goals are two-fold: (1) to infer G_{prod}, which is entirely unobserved, (2) to predict future transactions, i.e., future hyperedges in G_{txns}.

3.2 Model Architecture

To learn from TPGs, we introduce a new class of models that combine temporal GNNs with a novel inventory module to jointly learn production functions and predict future edges. First, we describe our inventory module, which can either operate as a stand-alone model or be attached to any GNN. Second, we describe extended versions of two popular temporal GNNs, Temporal Graph Network (Rossi et al. 2020) and GraphMixer (Cong et al. 2023), which we refer to as SC-TGN (shown in Figure 1b) and SC-GraphMixer, respectively, with SC standing for supply chain.

Inventory module. The basic idea of our inventory module is that it explicitly represents each firm's inventory of products, and it adds and subtracts from the inventory based on products bought and consumed, respectively. Let $\mathbf{x}_i^{(t)} \in \mathbb{R}_+^m$ represent firm i's inventory at time t. We compute the total amount of product p bought by firm i at time t as

$$\text{buy}(i, p, t) = \sum_{e(s,i,p,t) \in \mathcal{E}} \text{amt}(s, i, p, t), \tag{1}$$

where $\text{amt}(s, i, p, t)$ represents the amount of the product in the transaction indexed by s, i, p, t. The amount bought per product is computed directly from the data, since we observe the products that a firm buys through its transactions. On the other hand, we cannot observe the products that a firm *consumes* from their inventory; only the finished products that they supply to others. So, we need to learn the function mapping from products externally supplied to products internally consumed from the inventory. We estimate the total amount of product p consumed by firm i at time t as

$$\text{cons}(i, p, t) = \sum_{e(i,b,p_s,t) \in \mathcal{E}} \alpha_{p_s p} \cdot \text{amt}(i, b, p_s, t), \tag{2}$$

where $\alpha_{p_s p} \in \mathbb{R}_+$ is a learned attention weight representing how much of product p is needed to make one unit of the product supplied, p_s. In other words, each product *attends to* its parts. Finally, let $\mathbf{b}_i^{(t)}$ represent the vector over all products of amount bought by firm i at time t, and let $\mathbf{c}_i^{(t)}$ be defined analogously for consumption. Then the firm's updated inventory is

$$\mathbf{x}_i^{(t+1)} = \max(0, \mathbf{x}_i^{(t)} + \mathbf{b}_i^{(t)} - \mathbf{c}_i^{(t)}). \tag{3}$$

We take the elementwise max with 0 to ensure that the inventory stays non-negative, but we also penalize whenever consumption exceeds the current inventory amounts (5).

Attention weights. If the inventory module is standalone, then we directly learn the pairwise weights $\alpha_{p_1 p_2}$ for all product pairs p_1, p_2. If the inventory module has access to product embeddings (e.g., from a GNN), we can use product embedding $\mathbf{z}_p \in \mathbb{R}^d$ to inform the attention weights as

$$\alpha_{p_1 p_2} = \text{ReLU}(\mathbf{z}_{p_1} \mathbf{W}_{\text{att}} \mathbf{z}_{p_2} + \nu_{p_1 p_2}), \tag{4}$$

where $\mathbf{W}_{\text{att}} \in \mathbb{R}^{d \times d}$ and $\nu_{p_1 p_2} \in \mathbb{R}$ are learned parameters, and we apply ReLU to ensure that the attention weights are

[1] For simplicity, we assume in this work that there is one way to make each product, but future work may consider extensions, such as firm-specific product graphs or substitute products.

non-negative. Compared to directly learning the attention weights, here we treat $\mathbf{z}_{p_1}\mathbf{W}_{\text{att}}\mathbf{z}_{p_2}$ as the base rate and $\nu_{p_1p_2}$ as adjustments, which we encourage to be small in magnitude with L_2 regularization (6). By using the embeddings to form the base rate, instead of learning each pair independently, we can share information across product pairs, which is especially useful given sparse real-world data where we rarely observe most pairs.

Inventory loss. To train the inventory module, we introduce a special loss function. For a given firm i at time t, its inventory loss is

$$\ell_{\text{inv}}(i,t) = \lambda_{\text{debt}} \sum_{p\in[m]} \max(0, \text{cons}(i,p,t) - \mathbf{x}_i^{(t)}[p]) \quad (5)$$
$$- \lambda_{\text{cons}} \sum_{p\in[m]} \text{cons}(i,p,t).$$

That is, we penalize inventory debt, i.e., whenever consumption exceeds the current inventory, but otherwise reward consumption. We penalize inventory debt since firms should not be able to consume products that they never received. Furthermore, penalizing inventory debt results in *sparse* attention weights, since for most firms, we do not observe it buying most products, so for any of those products that it does not buy, it would prefer to never consume those products since it would immediately go into inventory debt if so. On the other hand, we need the consumption reward in order to prevent trivial solutions. Without it, the model could learn $\alpha_{p_1,p_2} \approx 0$ for all product pairs, and it would never experience inventory debt. We use hyperparameters λ_{debt} and λ_{cons} to control the relative weight between penalizing inventory debt and rewarding consumption, and in practice, we find that choosing λ_{debt} around 25% larger than λ_{cons} works well (Table 5). All together, the inventory loss is

$$\ell_{\text{inv}}(t) = \frac{1}{n}\sum_{i\in[n]} \ell_{\text{inv}}(i,t) + \lambda_{L_2}\sqrt{\sum_{p_1,p_2\in[m]} \nu_{p_1p_2}^2}. \quad (6)$$

SC-TGN. The first GNN we explore is Temporal Graph Network (TGN) (Rossi et al. 2020), which is one of the most established GNNs for dynamic link prediction, outperforming other models in the Temporal Graph Benchmark (Huang et al. 2023). In our work, we have extended TGN to SC-TGN, by enabling it to (1) perform message passing over *hypergraphs*, since we represent each transaction as an edge between three nodes, (2) predict edge weights (i.e., transaction amounts) in addition to edge existence. We also modified TGN in other ways that improved performance; we document these changes in Appendix A.1. In SC-TGN, each node i has a time-varying memory $\mathbf{m}_i^{(t)}$. Each transaction $e(s,b,p,t)$ sends three messages: to supplier s, buyer b, and product p. At the end of each timestep, each node aggregates the messages it received and updates its memory, using a recurrent neural network (RNN). To produce node embedding $\mathbf{z}_i^{(t)}$, we apply a GNN to the node memories, so that nodes can also learn from their neighbors' memories.

SC-GraphMixer. We also explore GraphMixer (Cong et al. 2023), a recent model that showed that temporal GNNs do not always need complicated architectures, such as RNNs or self-attention (both of which are used by TGN), and strong performance can sometimes be achieved with simpler models that only rely on multi-layer perceptrons (MLPs). We similarly extend GraphMixer to SC-GraphMixer, so that it can handle hypergraphs and predict edge weight in addition to edge existence (Appendix A.2). In SC-GraphMixer, each node's embedding $\mathbf{z}_i^{(t)}$ is a concatenation of its node encoding and link encoding. The node encoding is simply a sum of the node's features and mean-pooling over its 1-hop neighbors' features. The link encoding summarizes the recent edges that the node participated in, by applying an MLP to the time encodings and features of the recent edges.

Decoder. Both models, SC-TGN and SC-GraphMixer, use the same decoder architecture, and we use the same architecture (although separate decoders) for predicting edge existence and weight. We model these as a two-step process: first, predicting whether an edge exists; second, conditioned on the edge existing, predicting its weight. The decoder is a two-layer MLP over the concatenated supplier firm's, buyer firm's, and product's embeddings, producing $\hat{y} \in \mathbb{R}$:

$$\hat{y}(s,b,p,t) = \text{MLP}([\mathbf{z}_s^{(t)}|\mathbf{z}_b^{(t)}|\mathbf{z}_p^{(t)}]). \quad (7)$$

Real values are natural for both tasks, since for edge existence, we apply a softmax and evaluate the probability of the positive transaction compared to negative samples, and for edge weight, we preprocess the transaction amounts with log-scaling, so negative predicted amounts are valid.

When the inventory module is attached, we may also allow it to inform edge prediction. For edge existence, the inventory module *penalizes* impossible transactions, i.e., some (s,b,p,t) where supplier s does not have the parts required to make product p in its inventory at time t, and subtracts the penalty (14) from the model's original $\hat{y}$ (7). For edge weight, the inventory module computes the maximum amount of product p that supplier s could produce, based on its inventory, and *caps* $\hat{y}$ based on the computed maximum (15). Thus, as shown in Figure 1b, the inventory module and GNN help each other, with the GNN's embeddings informing the inventory module's attention weights, and the inventory module's penalties affecting future edge prediction.

3.3 Model Training and Evaluation

Learning production functions. Our model does not have access to any production functions during training, but the goal is for the model to learn production functions via the inventory module and its specialized loss. For each product p, we have the set of its attention weights $\{\alpha_{pp_1}, \alpha_{pp_2}, \cdots, \alpha_{pp_m}\}$, and we compute the ranking over all products from highest to lowest weight. We compare this ranking to the product's ground-truth parts and use average precision (17) to quantify performance. Then, we compute the mean average precision (MAP) over all products for which we have ground-truth parts.

Edge existence and negative sampling. We perform negative sampling such that each positive transaction $e(s,b,p,t)$

is paired with a set of negative transactions that did not actually occur at time t. Following Temporal Graph Benchmark (Huang et al. 2023), we sample two types of hard negatives: first, randomly perturbing one of the three nodes; second, sampling a *historical* negative, meaning a transaction that appeared in training but not at time t. For each positive transaction, we sample 9 perturbation negatives and 9 historical negatives. To evaluate performance, we use mean reciprocal rank (MRR), which evaluates the rank of the positive transaction among the negatives (21). However, MRR is non-differentiable, so during training, we use softmax cross-entropy as a strong proxy loss for MRR (Bruch et al. 2019).

Edge weight. Since we model edge prediction as a two-stage process, we only predict edge weight (i.e., transaction amount) conditioned on the edge existing. Thus, training and test edge weight prediction is simple: we only consider the positive transactions and we compare the model's predicted amount to the true amount using root mean squared error (RMSE) (22). Unlike MRR, RMSE is differentiable, so we also use it in the model loss during training.

4 Supply Chains Data

4.1 Real-World Supply Chains Data

We acquired transactions data from TradeSparq, a third-party data provider which aggregates data from authorized government sources across 60+ countries. Their data sources include bills of lading, receipts of reported transactions, and customs declarations. Each product is described with a Harmonized System (HS) code, an internationally recognized system for classifying products. Along with HS codes, the transactions data also includes the supplier firm, buyer firm, timestamp of the transaction, cost in USD, and more. Using the TradeSparq API, we constructed two datasets:

1. Our `Tesla` dataset focuses on electric vehicles (EV) and EV parts supplied by Tesla. We identified Tesla EV makers, their direct suppliers and buyers, and their suppliers' suppliers, and included all transactions between these firms from January 1, 2019, to December 31, 2022.
2. Our industrial equipment dataset (`IED`) focuses on microscopes along with other specialized analytical and inspection equipment and their manufacturers. We included makers of these products and their direct suppliers and buyers, and included all transactions between these firms in 2023. For this dataset, we have also estimated the ground-truth parts of microscopes in terms of their HS codes, which we use to test our inventory module's ability to infer production functions from transactions.

4.2 Supply Chains Simulator, `SupplySim`

The TradeSparq data offers a rare opportunity to test models on real transactions data, but it also has shortcomings: we cannot release the data, it does not include most production functions, and, even though it provides more detailed information than most supply chains datasets, its transactions are still incomplete (e.g., missing domestic transactions). Thus, we design a simulator, `SupplySim`, that addresses these

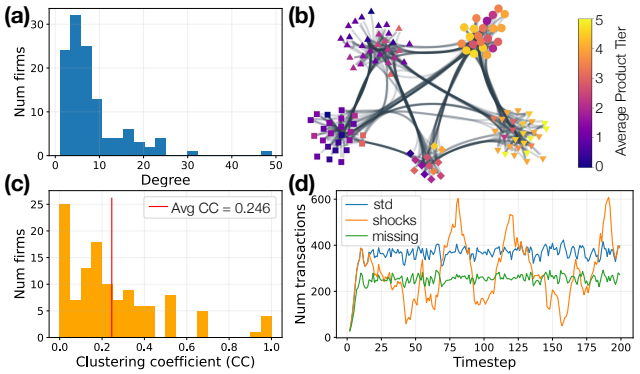

Figure 2: `SupplySim` generates data matching real data on key characteristics: **(a)** power law degree distribution, **(b)** community structure, **(c)** low clustering, **(d)** time-varying transactions, with possible shocks or missing data.

shortcomings and enables us to test the model under controlled settings (e.g., how much data is missing). To ensure realism, our simulator incorporates many real-world aspects of supply chains: for example, firms specialize in certain tiers of products and time-varying transactions are generated based on the commonly used ARIO model from economics (Hallegatte 2008), which describes how firms complete orders from buyers and place orders to suppliers. We also show that our synthetic data matches real supply chain networks on key characteristics (Figure 2). For example, there is community structure but fewer triangles (thus, lower average clustering coefficient) since, unlike social networks, a firm's supplier's supplier is unlikely to also be this firm's supplier (Fujiwara and Aoyama 2010; Zhao, Zuo, and Blackhurst 2019). Our synthetic data also exhibits power law degree distributions, known to appear in real networks, and time-varying transactions with possible shocks or missing data. Below, we briefly describe the steps of our simulator, with details in Appendix B.2 and B.3.

Constructing the production graph, G_{prod}. First, we partition the products into tiers (e.g., products 0-4 in tier 0, 5-14 in tier 1, 15-24 in tier 2, etc.) and sample a 2-dimensional position for each product from Uniform$(0, 1)$. For each tier, we assign the products in the tier to parts from the previous tier, with probability proportional to the inverse distance between their positions. For each part-product pair, we sample u_{io}, the number of units of part p_i needed to make one unit of product p_o. The tier structure imitates tiers in real supply chains, from raw materials in the first tier to consumer products in the final tier. The products' positions capture the product type, e.g., its industry, and assigning parts based on positions reflects how parts should be similar to their products and naturally results in commonly co-occurring parts.

Constructing supplier-buyer graph. For each firm, we also sample a 2-dimensional position from Uniform$(0, 1)$, and we restrict it to two consecutive tiers, meaning it can only produce products in those tiers. Then, for each product, we select its suppliers from the set of firms that are al-

	SS-std	SS-shocks	SS-missing	IED
Random baseline	0.124 (0.009)	0.124 (0.009)	0.124 (0.009)	0.060 (0.002)
Temporal correlations	0.745	0.653	0.706	0.128
PMI	0.602	0.602	0.606	0.175
node2vec	0.280	0.280	0.287	0.127
Inventory module (direct)	0.771 (0.005)	0.770 (0.006)	0.744 (0.006)	0.143 (0.004)
Inventory module (emb)	**0.790** (0.005)	**0.778** (0.011)	**0.755** (0.007)	**0.262** (0.005)

Table 1: Results for production learning, evaluated with mean average precision (MAP ↑). For the models with randomness, we report mean and standard deviation (in parentheses) over 10 seeds.

lowed to produce that product, again with probability proportional to inverse distance. Now, each firm has a set of products that it is supplying, which means, based on G_{prod}, we know which input parts it needs to buy. For each pair (b, p), where firm b needs to buy product p, we assign it to a supplier of p with probability proportional to the number of buyers that the supplier already has. This assignment mechanism, known as preferential attachment (Newman 2001), yields power law degree distributions (Figure 2a) known to appear in real supply chain networks (Fujiwara and Aoyama 2010; Zhao, Zuo, and Blackhurst 2019).

Generating transactions. We generate transactions based on the ARIO model, an agent-based model widely used in economics to simulate propagation over supply chains (Hallegatte 2008; Inoue and Todo 2019; Guan et al. 2020). At each timestep of the simulation, each firm completes as many of its incomplete orders as it can until it runs out of inventory. At the end of the timestep, the firm places orders to its suppliers, based on what it needs to complete its remaining orders. Finally, the reported transactions are the completed orders in each timestep. The simulator also keeps track of the exogenous supply for products in the first tier, which do not require parts, and of exogenous demand for products in the final tier, which are only bought by consumers and not by other firms. By manipulating the exogenous supply, we can model shocks in the supply chain.

5 Experiments

We run experiments on the two real supply chain datasets and three synthetic datasets from SupplySim: a standard setting with high supply ("SS-std"), a setting with shocks to supply ("SS-shocks"), and a setting with missing transactions ("SS-missing"), where we sampled 20% of firms uniformly at random and dropped all of their transactions (Figure 2d). In all experiments, we order the transactions chronologically and split them into train (the first 70%), validation (the following 15%), and test (the last 15%). Here we describe our results on these datasets, with additional experimental details and results in Appendix C.

5.1 Learning Production Functions

We try three baselines, which each compute scores for output product p_o and potential input p_i:

1. **Temporal correlations**: we expect that inputs and outputs are temporally correlated, so this method computes the maximum correlation, with possible lags, between the buying timeseries of p_i and supplying timeseries of p_o, averaged over all firms that buy p_i and supply p_o.
2. **Pointwise Mutual Information (PMI)**: we expect that input-output pairs appear with greater frequency in the firm-product graph,[2] so this method computes the probability that a firm buys p_i *and* supplies p_o, divided by the product of their individual probabilities (29).
3. **node2vec**: we expect that inputs are close to outputs, so this method computes the cosine similarity of p_i and p_o's node2vec embeddings from the firm-product graph.

The three baselines capture the usefulness of temporal information, 1-hop neighbors in the graph, and the entire graph, respectively. In contrast, our inventory module captures both temporal and structural information. We try two versions of the inventory module, one that learns the attention weights directly and one that uses product embeddings, as described in (4). We also report a random baseline, which produces uniform random rankings of parts for each product.

We summarize the production learning results in Table 1. We find that the inventory module significantly outperforms the baselines, with especially large margins on the real-world data (IED) and in the synthetic data when there are shocks in supply. In the standard synthetic data, when supply is plentiful, temporal correlations are a strong predictor since firms place orders for inputs, receive them promptly, then supply their own outputs shortly after. However, once there are shocks, the firm will receive inputs at different times, due to delays, and since they cannot produce their outputs until all inputs have arrived, the correlation in time of buying inputs and supplying outputs is seriously worsened. On the other hand, the inventory module is robust to such delays, since it does not rely on similarity in timeseries; simply that an input must go into the inventory *before* the output comes out. The inventory module is also remarkably robust to missing data: the MAP only drops by 3.5 points (4.4%) when we drop 20% of firms in the synthetic data. We also see that using product embeddings, instead of learning the attention weights directly, consistently helps the inventory module.

Figure 3 visualizes our results, showing that the inventory module effectively learns the true production functions. The inventory module also learns attention weights of similar magnitude as the true production functions, while the other baselines are not comparable on magnitude, only ranking.

[2]The firm-product graph is a static bipartite graph of firm and product nodes, where an edge between a firm and product indicates that the firm buys or supplies the product.

	SS-std	SS-shocks	SS-missing	Tesla	IED
Edgebank (binary)	0.174	0.173	0.175	0.131	0.164
Edgebank (count)	0.441	0.415	0.445	0.189	0.335
Static	0.439 (0.001)	0.392 (0.002)	0.442 (0.001)	0.321 (0.001)	0.358 (0.001)
Graph transformer	0.431 (0.003)	0.396 (0.024)	0.428 (0.003)	0.507 (0.020)	0.613 (0.045)
SC-TGN	0.522 (0.003)	0.449 (0.004)	**0.494** (0.004)	**0.820 (0.007)**	**0.842 (0.004)**
SC-TGN+inv	**0.540** (0.003)	**0.461** (0.009)	0.494 (0.004)	0.818 (0.004)	0.841 (0.008)
SC-GraphMixer	0.453 (0.005)	0.426 (0.004)	0.446 (0.003)	0.690 (0.027)	0.791 (0.009)
SC-GraphMixer+inv	0.497 (0.004)	0.448 (0.004)	0.446 (0.002)	0.681 (0.014)	0.791 (0.008)

Table 2: Results for predicting existence of future edges, evaluated with mean reciprocal rank (MRR ↑). We report mean and standard deviation (in parentheses) over 10 seeds. Edge weight results are provided in the Appendix (Table 4).

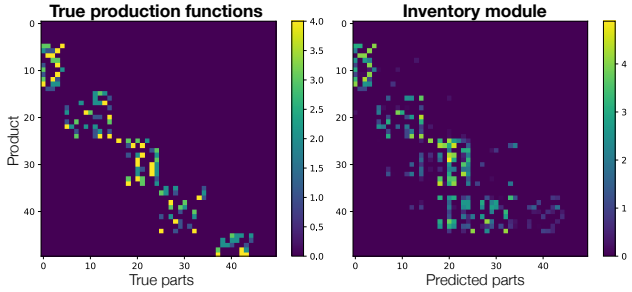

Figure 3: True production functions (left) and predictions from inventory module (right), trained on SS-std.

While only ranking matters for MAP, magnitude is essential if using the inventory module to inform edge prediction with penalties and caps. Finally, in Figure 5, we show that our inventory module's loss function (6) is well-correlated with MAP, which is why it can effectively learn production functions without observing any of them.

5.2 Predicting Future Edges

For edge existence, where we seek to predict whether future transaction (s, b, p, t) exists, we compare our models to the following baselines:

1. **Edgebank**: "binary" predicts 1 if $(s, b, p, *)$ appeared before in the train set; 0 otherwise. "count" predicts the number of times that $(s, b, p, *)$ appeared in the train set.
2. **Static**: learns a static vector to represent each node.
3. **Graph transformer**: learns a static embedding to represent each node, using the graph transformer model called UniMP from Shi et al. (2021).

While Edgebank simply memorizes the train set, it serves as a strong baseline, as the Temporal Graph Benchmark (Huang et al. 2023) found that even the binary version outperformed some GNNs on dynamic link prediction. The latter two baselines test two types of static node embeddings, which allow us to isolate the benefit of temporal GNNs. For each of our models (SC-TGN and SC-GraphMixer), we try them alone and with the inventory module (+inv).

In Table 2, we summarize results for predicting edge existence; results for predicting edge weight tell a very similar story (Table 4). First, we find that our models, SC-TGN and SC-GraphMixer, outperform the baselines on both tasks and all five datasets, by 11-62% on edge existence and 1-13% on edge weight. Second, we find that SC-TGN consistently outperforms SC-GraphMixer; we hypothesize this is because of SC-TGN's sophisticated updating of node memories, while SC-GraphMixer only captures changes over time through its link encoder, which encodes the features and timestamps of the node's recent edges (Appendix A.2). Supporting this hypothesis is the fact that the static methods perform poorly, demonstrating the need for temporal GNNs for supply chains. Since we find that SC-TGN outperforms SC-GraphMixer, we also compared SC-TGN to ablated versions: the original TGN, to test the value of our extensions (which we document in Appendix A.1), and to SC-TGN where the node memory is directly used as the embedding, instead of applying a GNN to the memories. We find that the full SC-TGN outperforms both ablations substantially, by 34-45% and 53-100%, respectively (6).

We also find that, in the synthetic data experiments, the setting with shocks is the hardest for all models. This demonstrates how shocks complicate prediction, since shocks both delay firms' abilities to complete their orders and limit them to producing smaller amounts, thus affecting both edge existence and edge weight. However, adding the inventory module significantly improves both SC-TGN and SC-GraphMixer's MRRs under shocks. In general, even though adding the inventory module introduces an additional loss so that we can learn production functions (16), we find that it does not hurt edge prediction performance, and in several cases, improves performance.

6 Discussion

In this work, we have formalized temporal production graphs (TPGs), with supply chains as a canonical example, and developed a new class of GNNs that can handle TPGs. We also release a new open-source simulator, SupplySim, which enables rigorous model testing and future research development in this area. Our models successfully achieve two essential objectives—inferring production functions and predicting future edges—while preexisting GNNs focused on the latter. Our models can be used in a wide variety of real-world scenarios, such as demand forecasting, early risk detection, and inventory optimization. To concretely illustrate these possibilities, we provide a case study in the extended version of our paper, where we show how the model can be used to predict which firms will be affected by an emerging supply chain disruption. In future work, we hope

to explore other potential use cases in supply chains, apply our model to TPGs in other domains, and develop theoretical results for our inventory module, such as establishing identifiability conditions and connecting it to causal inference.

Acknowledgements

S.C. was supported in part by an NSF Graduate Research Fellowship, the Meta PhD Fellowship, and NSF award CCF-1918940. J.L. was supported in part by NSF awards OAC-1835598, CCF-1918940, DMS-2327709, and Stanford Data Applications Initiative. The authors thank Emma Pierson and members of Jure Leskovec's lab for helpful comments on the draft. From Hitachi, we are thankful to Arnab Chakrabarti for his continuous support.

References

Acemoglu, D.; Carvalho, V. M.; Ozdaglar, A.; and Tahbaz-Salehi, A. 2012. The Network Origins of Aggregate Fluctuations. *Econometrica*, 80(5): 1977–2016.

Aigner, D. J.; and Chu, S. F. 1968. On Estimating the Industry Production Function. *The American Economic Review*, 58(4): 826–839.

Assaad, C. K.; Devijver, E.; and Gaussier, E. 2022. Survey and Evaluation of Causal Discovery Methods for Time Series. *Journal of Artificial Intelligence Research*, 73.

Aziz, A.; Kosasih, E. E.; Griffiths, R.-R.; and Brintrup, A. 2021. Data Considerations in Graph Representation Learning for Supply Chain Networks. In *ICML 2021 Workshop on Machine Learning for Data*.

Baryannis, G.; Dani, S.; and Antoniou, G. 2019. Predicting supply chain risks using machine learning: The trade-off between performance and interpretability. *Future Generation Computer Systems*, 101: 993–1004.

Baumgartner, T.; Malik, Y.; and Padhi, A. 2020. Reimagining industrial supply chains. *McKinsey & Company*.

Brintrup, A.; Pak, J.; Ratiney, D.; Pearce, T.; Wichmann, P.; Woodall, P.; and McFarlane, D. 2020. Supply chain data analytics for predicting supplier disruptions: a case study in complex asset manufacturing. *International Journal of Production Research*, 58(11): 3330–3341.

Bruch, S.; Wang, X.; Bendersky, M.; and Najork, M. 2019. An Analysis of the Softmax Cross Entropy Loss for Learning-to-Rank with Binary Relevance. In *Proceedings of the 2019 ACM SIGIR International Conference on the Theory of Information Retrieval (ICTIR'19)*.

Carvalho, V. M.; Nirei, M.; Saito, Y. U.; and Tahbaz-Salehi, A. 2021. Supply Chain Disruptions: Evidence from the Great East Japan Earthquake. *The Quarterly Journal of Economics*, 136(2): 1255–1321.

Carvalho, V. M.; and Tahbaz-Salehi, A. 2019. Production Networks: A Primer. *Annual Review of Economics*, 11: 635–663.

Chang, S.; Koehler, F.; Qu, Z.; Leskovec, J.; and Ugander, J. 2024. Inferring dynamic networks from marginals with iterative proportional fitting. In *Proceedings of the 41st International Conference on Machine Learning (ICML'24)*.

Chang, S.; Vrabac, D.; Leskovec, J.; and Ugander, J. 2023. Estimating geographic spillover effects of COVID-19 policies from large-scale mobility networks. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI'23)*.

Coelli, T. J.; Rao, D. S. P.; O'Donnell, C. J.; and Battese, G. E. 2005. *An Introduction to Efficiency and Productivity Analysis*. Springer Science & Business Media.

Cong, W.; Zhang, S.; Kang, J.; Yuan, B.; Wu, H.; Zhou, X.; Tong, H.; and Mahdavi, M. 2023. Do We Really Need Complicated Model Architectures For Temporal Networks? In *Proceedings of the 11th International Conference on Learning Representations (ICLR'23)*.

Fujiwara, Y.; and Aoyama, H. 2010. Large-scale structure of a nation-wide production network. *The European Physical Journal B*, 77: 565—-580.

Guan, D.; Wang, D.; Hallegatte, S.; Davis, S. J.; Huo, J.; Li, S.; Bai, Y.; Lei, T.; Xue, Q.; Coffman, D.; Cheng, D.; Chen, P.; Liang, X.; Xu, B.; Lu, X.; Wang, S.; Hubacek, K.; and Gong, P. 2020. Global supply-chain effects of COVID-19 control measures. *Nature Human Behaviour*, 4: 577–587.

Hallegatte, S. 2008. An Adaptive Regional Input-Output Model and its Application to the Assessment of the Economic Cost of Katrina. *Risk Analysis*, 28(3): 779–799.

Huang, S.; Poursafaei, F.; Danovitch, J.; Fey, M.; Hu, W.; Rossi, E.; Leskovec, J.; Bronstein, M.; Rabusseau, G.; and Rabbany, R. 2023. Temporal Graph Benchmark for Machine Learning on Temporal Graphs. In *Proceedings of the 37th Annual Conference on Neural Information Processing Systems (NeurIPS'23)*.

Inoue, H.; and Todo, Y. 2019. Firm-level propagation of shocks through supply-chain networks. *Nature Sustainability*, 2: 841–847.

Inoue, H.; and Todo, Y. 2020. The propagation of economic impacts through supply chains: The case of a megacity lockdown to prevent the spread of COVID-19. *PLoS ONE*, 15(9).

Jiang, W.; and Luo, J. 2022. Graph neural network for traffic forecasting: A survey. *Expert Systems with Applications: An International Journal*, 207(C).

Kosasih, E. E.; and Brintrup, A. 2021. A machine learning approach for predicting hidden links in supply chain with graph neural networks. *International Journal of Production Research*, 60(17): 5380–5393.

Kumar, R.; Tomkins, A.; Vassilvitskii, S.; and Vee, E. 2015. Inverting a Steady-State. In *Proceedings of the 8th ACM International Conference on Web Search and Data Mining (WSDM'15)*.

Li, Y.; Chen, K.; Collignon, S.; and Ivanov, D. 2021. Ripple effect in the supply chain network: Forward and backward disruption propagation, network health and firm vulnerability. *European Journal of Operational Research*, 291: 1117–1131.

Li, Y.; Zobel, C. W.; Seref, O.; and Chatfield, D. 2020. Network characteristics and supply chain resilience under conditions of risk propagation. *International Journal of Production Economics*, 223.

Liu, Z.; Wan, G.; Prakash, B. A.; Lau, M. S. Y.; and Jin, W. 2024. A Review of Graph Neural Networks in Epidemic Modeling. In *arXiv*.

Longa, A.; Lachi, V.; Santin, G.; Bianchini, M.; Lepri, B.; Liò, P.; Scarselli, F.; and Passerin, A. 2023. Graph Neural Networks for temporal graphs: State of the art, open challenges, and opportunities. In *arXiv*.

Löwe, S.; Madras, D.; Zemel, R.; and Welling, M. 2022. Amortized Causal Discovery: Learning to Infer Causal Graphs from Time-Series Data. In *Proceedings of the 1st Conference on Causal Learning and Reasoning (CLeaR'22)*.

Maystre, L.; and Grossglauser, M. 2017. ChoiceRank: Identifying Preferences from Node Traffic in Networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*.

Nauta, M.; Bucur, D.; and Seifert, C. 2019. Causal Discovery with Attention-Based Convolutional Neural Networks. *Machine Learning and Knowledge Extraction*, 1(1): 312–340.

Newman, M. E. J. 2001. Clustering and preferential attachment in growing networks. *Physical Review E*, 64.

Prill, R. J.; Iglesias, P. A.; and Levchenko, A. 2005. Dynamic Properties of Network Motifs Contribute to Biological Network Organization. *PLoS Biology*, 3(11).

Rossi, E.; Chamberlain, B.; Frasca, F.; Eynard, D.; Monti, F.; and Bronstein, M. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. In *ICML 2020 Workshop on Graph Representation Learning*.

Shi, Y.; Huang, Z.; Feng, S.; Zhong, H.; Wang, W.; and Sun, Y. 2021. Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI'21)*.

Skarding, J.; Gabrys, B.; and Musial, K. 2021. Foundations and Modeling of Dynamic Networks Using Dynamic Graph Neural Networks: A Survey. *IEEE Access*, 9: 79143–79168.

The White House. 2021. Executive Order on America's Supply Chains. Available at https://www.whitehouse.gov/briefing-room/presidential-actions/2021/02/24/executive-order-on-americas-supply-chains/.

Tu, Y.; Li, W.; Song, X.; Gong, K.; Liu, L.; Qin, Y.; Liu, Y.; and Liu, M. 2024. Using graph neural network to conduct supplier recommendation based on large-scale supply chain. *International Journal of Production Research*, 1–14.

Wang, R.; Walters, R.; and Yu, R. 2021. Incorporating Symmetry into Deep Dynamics Models for Improved Generalization. In *Proceedings of the 9th International Conference on Learning Representations (ICLR'21)*.

Wasi, A. T.; Islam, M. S.; and Akib, A. R. 2024. SupplyGraph: A Benchmark Dataset for Supply Chain Planning using Graph Neural Networks. *arXiv*.

Wu, D.; Wang, Q.; and Olson, D. L. 2023. Industry classification based on supply chain network information using Graph Neural Networks. *Applied Soft Computing*, 132.

Younis, H.; Sundarakani, B.; and Alsharairi, M. 2022. Applications of artificial intelligence and machine learning within supply chains:systematic review and future research directions. *Journal of Modelling in Management*.

Zhao, K.; Zuo, Z.; and Blackhurst, J. V. 2019. Modelling supply chain adaptation for disruptions: An empirically grounded complex adaptive systems approach. *Journal of Operations Management*, 65(2): 190–212.