Proceedings of the 2025 Winter Simulation Conference
E. Azar, A. Djanatliev, A. Harper, C. Kogler, V. Ramamohan, A. Anagnostou, and S. J. E. Taylor, eds.

WHEN INTERPRETABILITY MEETS EFFICIENCY: INTEGRATING EMULATION INTO SUPPLY CHAIN SIMULATION

Xiaoyu Lu[1], Yujing Lin[2], Hoiyi Ng[3], and Yunan Liu[4]

[1]Amazon SCOT, Bellevue, WA, USA
[2]Amazon SCOT, Austin, TX, USA
[3,4]Amazon SCOT, New York City, NY, USA

ABSTRACT

For large-scale retail businesses such as Amazon and Walmart, simulation is critical for forecasting inventory, planning, and decision-making. Traditional digital-twin simulators, which are powered by complex optimization algorithms, are high-fidelity but can be computationally expensive and difficult to experiment with. We propose a hybrid simulator called SEmulate that integrates machine learning and emulation into large-scale simulation systems that follows the causal relationship between system components. SEmulate uses machine learning to emulate complex high-dimensional system components, while using physics-based modeling or heuristics for other simpler components. We apply SEmulate on supply chain simulation, where we demonstrate that SEmulate reduces runtime by orders of magnitude compared to the traditional digital-twin based simulator, while maintaining competitive accuracy and interpretability. The enhanced simulator enables efficient supply chain simulation, rapid experimentation, faster and improved decision-making process.

1 INTRODUCTION

In today's hypercompetitive business environment, where companies like Amazon and Walmart process millions of packages daily, the complexity of supply chain management has reached unprecedented levels (Acimovic and Graves 2015; Acimovic and Farias 2019; Amazon 2023; Liu et al. 2023). Supply chain simulation has emerged as a critical strategic tool, enabling organizations to navigate the intricate web of global logistics, inventory management, and customer fulfillment. This technological evolution in supply chain management represents not merely an operational enhancement but a fundamental shift in how organizations conceptualize and optimize their supply networks.

Through advanced simulation techniques, organizations can create digital replicas of their entire supply chain ecosystem, enabling them to test scenarios, predict outcomes, and optimize operations without the risks and costs associated with real-world experimentation. The critical role of simulation in contemporary supply chain management manifests in several key dimensions. First, it enables predictive analytics and risk mitigation, allowing organizations to anticipate and prepare for potential disruptions - a capability that proved invaluable during the global supply chain crisis of 2020-2022. Second, it facilitates real-time decision-making through dynamic modeling of complex variables, including demand fluctuations, inventory levels, and transportation networks. Third, it supports strategic planning by enabling organizations to evaluate the impact of major decisions, such as facility location or network redesign, before committing substantial resources.

Discrete-event simulation (DES) (Matloff 2008; Robinson 2014) is an *operations research* (OR) methodology that models the operation of a system as a sequence of discrete events occurring at specific points in time, where event marks a change of state in the system. It is one of the most common simulation frameworks in the industry. An inventory management system typically involve multiple complex systems

interacting with each other, a traditional digital-twin based simulator integrates all the production system components together, resulting in high fidelity. However, it has several drawbacks: 1) the integration efforts typically involve months of development effort; 2) it is difficult to maintain and catch up with all the system changes; 3) the runtime of the simulator is slow (e.g., can take several hours or days), and scales with the complexity of the systems (Farias et al. 2025). This causes delay in decision making and can have deteriorating downstream impact when the runtime exceeds the business timeline.

Emulation, on the other hand, is a technique designed to mimic the behavior and functionality of another system through models without replicating the system. Emulation has a long existing history in science and engineering for testing and understanding systems in which physical experimentation is infeasible or expensive (Tucker 1965; Young and Ratto 2011; Grow and Hilton 2014). Common emulators typically include building surrogate *machine learning (ML)* models (Mitchell and Mitchell 1997; Jordan and Mitchell 2015) such as *Gaussian processes* (GP) (Williams and Rasmussen 1995; Schulz et al. 2018), or with more recent advances in *deep learning* (DL) models (LeCun et al. 2015; Goodfellow et al. 2016).

Simulation and emulation are techniques that originate from separate fields: OR and ML, each rooted in fundamentally different philosophies. In OR, the focus is on understanding and optimizing the detailed dynamics of a system. Beyond simply solving a problem, OR seeks to uncover the underlying principles that govern it, providing structural insights into how the system works and how it can be improved. In contrast, ML prioritizes delivering high-quality solutions with a strong emphasis on end results. While ML models can achieve remarkable performance, they often do so by focusing on outcomes rather than intermediate steps or interpretability, with its internal mechanisms remaining opaque.

In other words, emulators can be regarded as efficient statistical representations of simulators. Unlike simulation, which replicates the detailed dynamics of a system, emulation learns the functional relationship of the system using surrogate models, which comes with the advantage of sensitivity analysis, uncertainty quantification and fast experimentation. However, one of the biggest drawback of emulators is that it lacks explainability especially for black-box models, and is hard to earn trust with stakeholders on the reliability of the models.

In this paper, we propose a hybrid framework that integrates emulation into simulation. We introduce an efficient and user-friendly simulator called Simulate & Emulate (SEmulate) which leverages machine learning methods to emulate complex system components, while also incorporating the underlying mechanisms of the simulation systems to preserve interpretability. This framework is generalizable to a broad range of applications such as queueing, aviation, and manufacturing. For this paper, we focus on an example application in supply chain management. We demonstrate that SEmulate can effectively mimic supply chain systems, while also adhering to the physics of supply chains to offer insights and interpretability into the simulated inventory dynamics. SEmulate learns the underlying system behaviors based on historical and/or simulated data, using ML models to capture complex patterns while incorporating the fundamental physics of inventory flow. This results in an explainable simulator capable of fast inventory flow prediction. This lightweight simulator offers flexibility for experimentation on supply chain systems, facilitating rapid decision-making and more efficient supply chain management, enabling a range of real-world applications such as forecasting, counterfactual impact estimation and input sensitivity analysis.

The paper is organized as follows: in Section 2 we give an overview of the high-level architecture of SEmulate; in Section 3 we discuss the methodology and the underlying models in each module in detail with an illustrative example; in Section 4 we apply SEmulate on a range of synthetic and real-world applications; finally we conclude in Section 5.

2 OVERVIEW OF SEMULATE ON RETAIL SUPPLY CHAIN SYSTEMS

SEmulate is a user-friendly simulator that leverages a hybrid approach to simulate complex systems. The key innovation lies in its ability to seamlessly integrate domain knowledge and causal relationships with data-driven machine learning techniques into simulation. The core steps in the SEmulate approach are:

1. **Establish causal relationships**: Construct causal relationships of the target systems by leveraging domain knowledge.
2. **Decompose system into sub-components**: Break down the overall system into different sub-components based on the identified causal linkages and underlying mechanisms. This modular design allows the simulator to better capture the intricate interdependencies within the system.
3. **Determine appropriate simulation Approach**: Decide how to simulate each individual sub-component:
   (a) For computationally expensive or unknown components, employ machine learning-based emulation or predictive models to mimic the "black box" behavior.
   (b) For components with known mechanisms or relationships, leverage this domain knowledge and simulate those parts directly.
   (c) Selectively apply the appropriate simulation approach for each sub-component to strike a balance between computational efficiency and fidelity to the system's true dynamics.
4. **Integrate heterogeneous simulation approaches**: Integrate these heterogeneous simulation approaches into a cohesive and scalable platform. SEmulate accomplishes this by carefully coordinating the data flow and causal interactions between the various sub-components, ensuring that the overall system behavior accurately reflects the underlying domain knowledge and empirical observations.
5. **Perform downstream tasks**: Perform downstream tasks, such as conducting counterfactual estimation or experimentation, by perturbing the causal graph and propagating the effects through the simulation flow.

By combining the strengths of both traditional simulation techniques and data-driven machine learning models, SEmulate offers a flexible and powerful desktop simulator that can tackle a wide range of complex systems. As an illustrative example, we apply SEmulate to the retail supply chain simulation to illustrate its usage. We first give a brief overview of an example of retail supply chain simulation system following the *MiniSCOT* (Amazon 2021) simulator: it begins with strategic order placing decisions where the system analyzes demand forecasts, inventory positions, and sales patterns across different retail locations to determine the optimal purchase order quantities and inbound warehouses. These decisions incorporate factors such as seasonal demands, promotional events, inventory holding costs, and vendor constraints to generate purchase orders that specify order quantities and destination warehouses, while adhering to the network and warehouse capacity (Maggiar et al. 2022). For large-scale retail businesses, the automated ordering systems is typically one of the most computationally intensive optimization algorithms, due to the multi-echelon planning nature spanning millions of products across hundreds to thousands of warehouse nodes (Farias et al. 2025). Once these purchasing decisions are made, the procurement process places the purchase orders to vendors, who determine whether to (partially) accept or reject the purchase orders, and ship the inventory into the retailer's fulfilment network for accepted orders. Inbound process handles the inventory shipped from vendors into the warehouses, while incorporating vendors' lead time (i.e., the time it takes to ship inventory from vendors to warehouses). The outbound phase involves shipping inventory out of the retailer's fulfilment network to meet end customers' demand.

Following the design of SEmulate, we first construct the causal relationship of the aforementioned supply chain as shown in Figure 1. Based on the causal graph, we categorize the components into *order placing*, *procurement*, *inbound* and *outbound*, and we choose the corresponding emulation/simulation method according to the underlying mechanism in each component.

1. **Order placing module**: Order placing is one of the most critical components in supply chain systems, it decides for each product, how many units we should buy into the warehouses. The ordering system is a high-dimensional, large-scale multi-echelon stochastic optimization problem in face of random demand and lead time, that is computationally very expensive. It first solves the optimal ideal inventory level known as the *target inventory position* (TIP) for each product across the warehouses. Given the supply, transfer cost, and capacity constraints in the network, it then

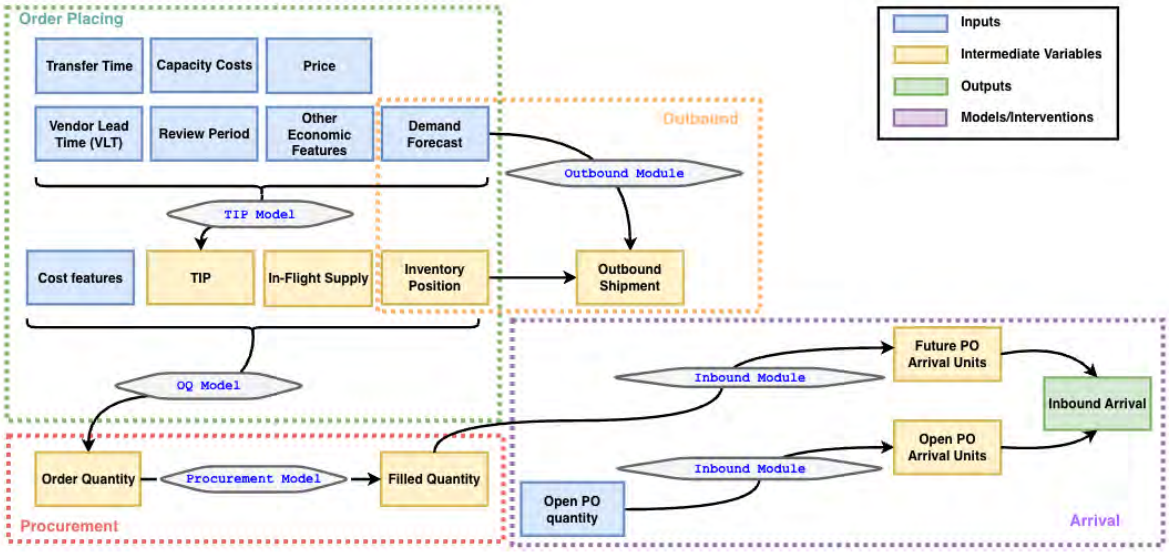

Figure 1: High-level logic flow of SEmulate. Each component is represented either by an emulator or a model designed according to the physical dynamics of the supply chain.

solves another high-dimensional, large-scale optimization problem to obtain the optimal ordering quantities, which we term as the *ordering quantity* (OQ). In the forward-looking simulation system, we typically simulate T weeks into the future, therefore the computation complexity grows linearly with $\mathscr{O}(T)$, making it the biggest computation bottleneck in simulation. To reduce computational complexity, we choose to emulate this module with ML models.

2. **Procurement module**: After we submit *purchase orders* (POs) to vendors, vendors decide whether to (partially) accept or reject the orders. This is outside of the systems' control with unknown mechanism, hence we built a ML prediction model to predict vendor's fill rate. The output of the module is the expected order quantity the vendors will ship to us, which we phrase as the *filled quantity* (FQ).
3. **Inbound module**: Given the predicted FQ, vendors ship the inventory to the warehouses. Assuming a known *vendor lead time* (VLT) distribution, the arrival process is a relatively simple process with known mechanism, hence we simulate it directly. In particular, we sample from the VLT distribution to simulate the arrival time of the FQ. The output of this module is the simulated PO inbound arrival, meaning how many units of the simulated POs arrive in each of the week over the next few weeks' horizon. In addition to the newly simulated POs, we also include the existing POs in the real world that are expected to arrive, which we phrase as *open PO*. Similar to the inbound process for simulated PO, we sample from the VLT distribution to simulate the arrival time of open POs. The sum of the open PO inbound and the simulated PO inbound gives the total inbound arrival units.
4. **Outbound module**: The outbound module refers to the process where inventories flow out of the fulfillment network when we ship inventories to the end customers. Assuming a known demand forecast distribution, it is a relatively simple process with known mechanism, hence we simulate the outbound process with simplified physics which depends on the customer demand forecast and available inventory in the warehouses.
5. **Supply Update Module**: This module dynamically update the supply states that include the status of the in-flight POs and available inventory after all of the above modules at each timestep t. The updated supply states are then fed into the order placing module at the next timestep $t+1$.

We iterate through this cycle for each timestep t into the future weeks, so that the evolution of each product's key supply chain metrics are propagated to the next computation cycle to generate predictions, providing an explainable view of inventory evolution over time. The pseudo code can be found in Algorithm 1, where we denote s_t as the set of all the supply chain metrics at time t which contains both the inputs and outputs. Note how some modules only impact the current time t (e.g., Procurement only impacts the current FQ) whereas some impact all future states (e.g., Inbound can impact inventory inbound in the future). This minimal formulation enables rapid, explainable experimentation on supply chain interventions, capturing key phenomena in inventory evolution. This light-weight hybrid simulator can easily run on desktops, offering a rapid and quick experimentation platform for users to test out different interventions.

Algorithm 1 SEmulate

Input: Input features specified in Figure 1, highlighted in blue. Pre-trained ML models. Simulation horizon (total number of timesteps) T.

Output: Intermediate/output features specified in Figure 1, highlighted in yellow/green.

```
for t ← 1 to T do
    s_t ← OrderPlacing(s_{t:T})
    s_t ← Procurement(s_t)
    s_{t:T} ← Inbound(s_{t:T})
    s_t ← Outbound(s_t)
    s_t ← SupplyUpdate(s_t)
end for
return s_{0:T}
```

3 METHODOLOGY ON INTEGRATING EMULATION INTO SIMULATION

In this section, we give more technical details for each of the components mentioned in Section 2. We particularly emphasize the methodology in the Order Placing module given its biggest computation challenge.

3.1 Order Placing Module

Traditionally, the ordering algorithms are OR models that optimizes the decision variables (e.g., TIP and OQ) subject to constraints (Hadley and Whitin 1963), and can be computationally costly (Maggiar et al. 2022). The novelty of SEmulate lies in that instead of solving the complex optimization, we utilize existing data and leverage ML to learn the relationship between the input features and the optimized decisions. Following the production system's mechanism, we break the *order placing* module into two pieces: TIP model f_{TIP} and OQ model f_{OQ} respectively. For model selection on the form of f_{TIP} and f_{OQ}, SEmulate is agnostic to the specific modeling choice. We provide a few default models and illustrate with one example of the model selection process. For the Amazon use cases, we choose the model based on three different perspectives: 1) accuracy; 2) explainability and 3) speed. The accuracy metric measures the predictive power of the model; the explainability ensures that the learned input-output relationship aligns with the business intuition. To elaborate, we know with domain knowledge how the inputs of the TIP model affect TIP directionally: for example, higher demand forecast should lead to higher TIP, higher capacity cost should lead to lower TIP, etc.. Hence an explainable model should reflect such monotonic relationship, which is critical for downstream experimentation reliability. Finally, satisfactory speed metric ensures that the simulator runs fast enough to meet the business timeline.

We provide a few default model choices, such as (monotonic) linear regression, (monotonic) tree-based models (e.g., XgBoost) and an additive Gaussian Process model (Lu et al. 2022) with summary of their attributes in Table 1. Users can specify their desired choice according to different requirements and specific applications. For both the TIP and OQ models, we give example input features, but the method is generalizable to any available set of input features.

Table 1: Model comparison on the order placing modules with a few default modeling choices.

Model	Accuracy	Explainability	Speed
Monotonic Linear Model	low	high	highest
XgBoost	high	low	high
Monotonic XgBoost	high	high	high
Additive Gaussian Process	high	high	low

3.1.1 TIP Model

We formulate TIP modeling into a supervised learning problem where the output variable TIP is a function of the input features:

$$TIP_t = f_{TIP}(\{\mathbf{D}_t^h\}_{h=1}^{h_{max}}, \{C_t^h\}_{h=1}^{h_{max}}, \mathbf{L}, TT, RP, p, \tilde{p}), \tag{1}$$

where $\mathbf{D}_t^h$ denotes the quantiles of the demand forecast distribution at time t with span h, $\mathbf{C_t}^h$ denotes the capacity cost at time t with span h, h_{max} denotes the maximum span, $\mathbf{L}$ denotes the quantiles from the vendor lead time distribution, TT denotes the transfer time between inbound dock to storage warehouses, RP denotes the review period, p denotes the product price and $\tilde{p}$ denotes other economic features. Note that $\mathbf{D}_t^h$ and C_t^h are time-dependent as we incorporate the seasonality of the demand forecast and time-varying capacity constraints. For our use case, we balance the three evaluation metrics in Table 1 and choose the monotonic XgBoost model with monotone constraints $[\{\mathbf{1}^h\}_h, \{-\mathbf{1}^h\}_h, \mathbf{1}, 1, 1, 1, 1, 1]$ where 1 represents monotonically increasing relationship whereas -1 represents monotonically decreasing relationship. Bold font represents vectors.

3.1.2 OQ Model

With the same model selection criteria, we choose to fit a monotonic XgBoost model for the OQ in a similar way with relevant features:

$$OQ_t = f_{OQ}(TIP_t, i_t, s_t, c_t), \tag{2}$$

where TIP_t denotes the target inventory position at time t, i_t denotes the inventory position at time t, s_t denotes the in-flight supply (PO) at time t and c_t denotes the cost-related features at time t, with monotonic constraint $[1, -1, -1, -1]$. At inference time, we use the TIP model to predict TIP_t which is then fed into the OQ model, the supply features such as the on-hand inventory i_t and in-flight supply s_i are dynamically updated and fed into the OQ prediction through time. In addition, we perform clustering pre-processing step to segment different products into different buckets based on their volume. This alleviates the heavy-tail problem where more popular products and less popular products exhibits different distribution and buying behaviors. We found the segmentation step significantly improves the model accuracy empirically.

3.2 Procurement Model

We aim to answer the question: "Given x units of OQ for each product, how many units will the vendors ship to us in total?" Since this is an unknown mechanism that is outside of the systems' control, we choose to build machine learning predictive models and we hereby give an example of such predictive model using a time series approach. Suppose at test time t, there is a predicted OQ of OQ_t, we first predict a fill rate r_t which we then multiple by OQ_t to get the predicted FQ_t for each product. We leverage features such as historical fill rate, order quantity and seasonality (one can customize special holidays such as Thanksgiving period etc. into this 1-hot encoding feature):

$$r_t = f_{fr}(r_{t-1}, r_{t-2}, \cdots, r_{t-\tau}, \log(OQ_t), \mathbf{M_t}, \mathbf{z}), \quad FQ_t = OQ_t \times r_t, \tag{3}$$

where $r_{t-1}, r_{t-2}, \cdots, r_{t-\tau}$ denotes the historical fill rates observed in the previous τ orders, OQ_t denotes the OQ at time t, where due to the heavy-tailed nature of OQ, we take $\log(OQ)$ as the feature, $\mathbf{M_t}$ denotes seasonality feature at time t, e.g., the month index using a one-hot encoding, $\mathbf{z}$ denotes the vector of additional vendor features. Similarly to the model selection process in the *order placing* module, the choice of f_{fr} is not unique, we found that both XgBoost and linear model achieve satisfactory explainability without additional constraints.

3.3 Inbound and Outbound Modules

We follow the physics of the supply chain to generate the arrival inbound of each PO. To account for the stochastic nature of the inbound process due to randomness in the time it takes for vendors to ship to us, and the fact that a single PO can be split into multiple shipments arriving in different weeks, we leverage Monte Carlo method to obtain an estimation of the arrival pattern. Specifically, for each predicted FQ (or existing PO), we sample n VLT: L_t^j for $j = 1, \cdots, n$ from the VLT distribution, after which we take the average of the inbound arrival patterns for each product:

$$q_t^j \sim Uniform(\mathbf{Q}) \; for \; j = 1, \cdots, n, \qquad L_t^j = \mathbf{Q}_{q_t^j} \; for \; j = 1, \cdots, n,$$
$$y^j_{t+[L_t^j]} = y^j_{t+[L_t^j]} + FQ_t \; for \; j = 1, \cdots, n, \qquad y_s = \frac{1}{n}\sum_{j=1}^{n} y_s^j \; for \; s >= t. \tag{4}$$

where $\mathbf{Q}$ denotes the quantiles of the VLT distribution, L_t^j denotes the sampled vendor lead time at time t for Monte Carlo trial j, $[\cdot]$ denotes the floor integer part of a float number, $y^j_{t+[L_t^j]}$ denotes the inbound arrival units at time $t + [L_t^j]$ for trial j, y_s denotes the average inbound arrival units at time s across n trials.

We model the outbound shipments for each product given the customer demand and available supply. Specifically, we first sample a customer demand units from the demand forecast distribution $\mathbf{D}_t$, e.g., with span $h = 1$ week, which represents weekly customer demands. We then simulate outbound shipment o_t accordingly conditioned on the available inventory for each product:

$$d_t \sim Uniform(\mathbf{D}_t^1), \quad o_t = \min(d_t, i_t). \tag{5}$$

Finally we update the available inventory i_{t+1} and in-flight PO s_{t+1} to be used for the next timestep $t+1$.

3.4 Putting it All Together

For machine learning *order placing* modules (TIP & OQ models), we train the models on synthetic data that is simulated according to the news vendor buying policy from MiniSCOT. Test R-squared for both TIP and OQ models are above 95%. In practice, the data we train the models upon are generated with complex, computationally expensive order placing systems, where fitting a machine learning model reduces the runtime substantially. We use a single-product anecdote to illustrate how SEmulate works end-to-end with pre-trained models: we first sample synthetic input data to feed into SEmulate for inventory simulation, where we assume the key inputs are generated in the following way: demand forecast: Gamma distribution with a linear drift for each week: $D_t \sim Gamma(200, 5) + 15t$, VLT distribution: $L_t \sim Uniform([3, 40])$, price: \$100, capacity cost: \$0, other economic feature: \$95, initial inventory: 500 units, initial in-flight PO: 100 units, initial in-flight PO arrival pattern: 50%, 30% and 20% in the first 3 weeks respectively. The key simulated metrics can be found in Figure 2. For example, on 2025-01-19, VLT = 7 days was sampled, meaning that the FQ of 3500 units in 2025-01-19 will arrive one week later in the week of 2025-01-26 as shown in the simulated inbound arrival plot. For outbound, we drain customer demand by sampling from demand forecast distribution and update the inventory states to be used in the next weekly computation cycle: in the week of 2025-01-26, the customer outbound shows 0 because there is no inventory available for sale.

Figure 2: Single-product illustration of SEmulate.

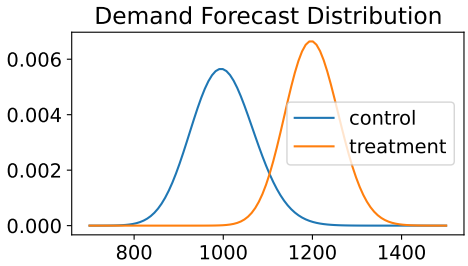

Figure 3: Demand forecast intervention.

4 APPLICATIONS

In this section, we demonstrate SEmulate's applications on both synthetic and real-world examples. To begin, we demonstrate SEmulate's capability to quantify the sensitivity of its output to changes in its input features.

4.1 Demand Forecast Counterfactual Estimation

Customer demand forecast is one of the most important inputs that affects order placing decisions and the customer outbound. How much a retailer order from vendors depend heavily on the end-customers' upcoming demand. A small lift in the demand forecast can lead to a large increase in the ordering, potentially disturbing the capacity management. Being able to estimate the impact of demand forecast is critical for the supply chain operation, so that the business can adjust labour planning and capacity control accordingly to prepare in advance for large increase in inventory inbound.

As an illustration, suppose the demand forecast distribution changes from $Gamma(200,5)$ (control group) to a different distribution $Gamma(400,3)$ (treatment group) as shown in Figure 4, we apply SEmulate

with both the original Gamma in the control group and the intervened Gamma in the treatment group to obtain the corresponding inventory impact for the total arrival units. We highlight that SEmulate follows the causal relationship of the supply chain system where both the order placing decisions and the outbound customer shipment are affected. Figure 4 shows the impact on TIP, outbound customer shipment, OQ and the total arrival. Aligning with our business intuition, a lift in the demand forecast leads to increase in the ordering and the incoming arrival volume, as well as the outbound shipment.

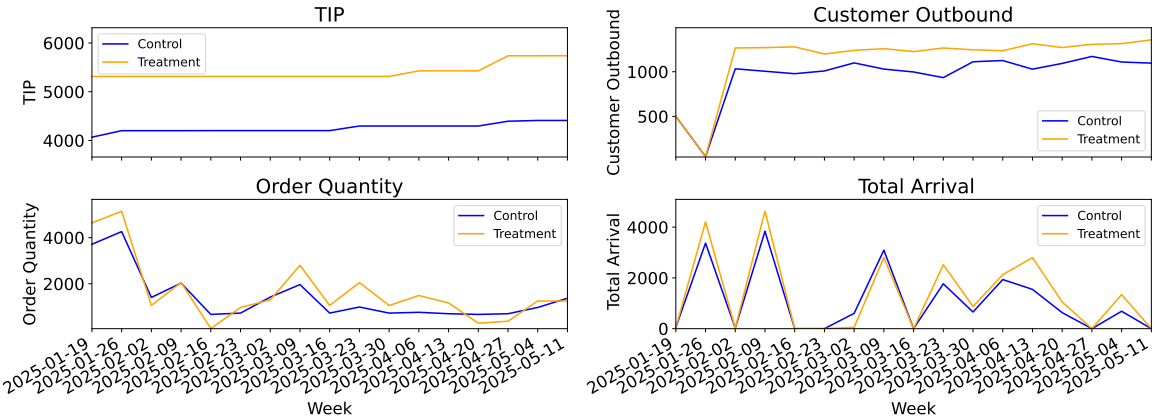

Figure 4: Demand forecast intervention counterfactual estimation.

4.2 VLT Counterfactual Estimation

VLT is another key input that impacts the supply chain systems. Similar to the demand forecast, it also affects multiple supply chain systems: 1) order placing decisions are affected, typically the longer the vendor lead time, the more orders are placed for the longer weeks of cover; 2) the arrival time of the POs is affected.

As an illustration, suppose the VLT distribution shifts by 5 days from $Uniform([3,40])$ (control group) to $Uniform([8,45])$ (treatment group), we apply SEmulate with both the original VLT in the control group and the intervened VLT in the treatment group to get the corresponding inventory impact for the total arrival units. Figure 5 shows the impact on TIP, sampled VLT, OQ and the total arrival. We observe that TIP increases as expected as higher inventory position is needed to account for the longer vendor lead time. We also observe a lift in ordering and arrival units initially, and a shift to later weeks in arrival due to increased sampled lead time.

4.3 Inbound Model Comparison

In addition to quantifying the downstream supply chain impact due to input changes, SEmulate can also be used to test certain system features or models. For example, one may be interested in testing the impact of the number of Monte Carlo samples n in equation (4). Specifically, for the control group: sample $n = 1$ single VLT from the uniform VLT distribution, then generate the arrival pattern with the arrival module $f_{arrival}$. For the treatment group: sample $n = 10$ VLTs uniformly and simulate $n = 10$ arrival times, then take the average of the arrival quantities for each week. The resulting impact are displayed in Figure 6. We observe much smoother ordering and arrival pattern due to increased number of Monte Carlo samples of VLT.

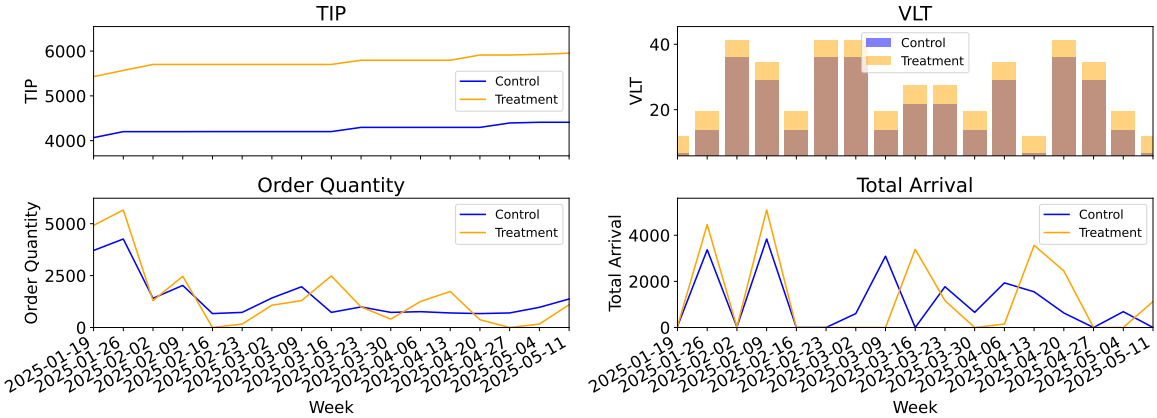

Figure 5: VLT distribution intervention counterfactual estimation.

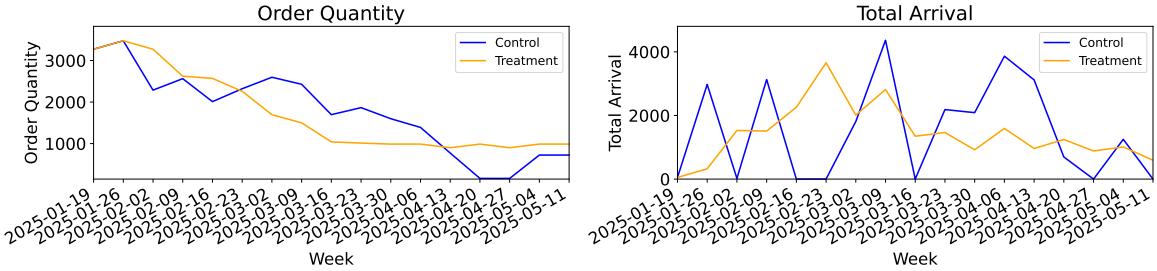

Figure 6: VLT Monte Carlo sampling counterfactual estimation.

4.4 System Intervention

In addition to testing the models behind supply chain systems, one can also test certain system interventions. For example, large retail businesses typically have certain guardrails to control the number of ordering, to prevent the order placing system from ordering too much or too little, either way will cause undesirable effect on the supply chain operation. As an illustrative example, suppose there is a default upper guardrail on OQ, such that it cannot exceed a certain threshold M. Intervening on the guardrail will impact the ordering behavior. Suppose we change M from unconstrained units to 3000 units, its counterfactual impact can be found in Figure 7. We observe that enabling the constraints in ordering leads to lower total arrival units in early weeks. The decreased ordering and inventory level in later weeks results in higher ordering in middle weeks, which further leads to no ordering in the last few weeks.

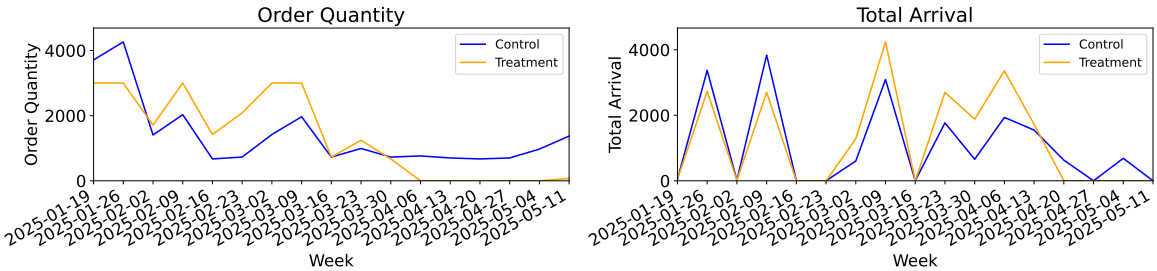

Figure 7: System intervention counterfactual estimation.

4.5 Amazon's Supply Chain Simulation

We apply SEmulate on a real-world application to enhance Amazon's retail supply chain simulation system. In particular, the baseline Amazon simulation system is a complicated OR based system that integrates multiple Amazon's production systems, resulting in a high-fidelity, low-efficiency simulator. We evaluate the performance of SEmulate from three perspectives: 1) accuracy and 2) efficiency and 3) capability with performance summary shown in Table 2. For 1), since our business use case cares most about the weekly aggregated inbound arrival volume across millions of products, we compare the simulated inbound inventory volume against the real world volume for both the baseline simulator and SEmulate, where we do not observe tangible degradation on accuracy. For 2), we compare the runtime of the two simulators. We observe SEmulate achieves significant improvement in efficiency with orders of magnitude reduction in runtime. In addition to improving the efficiency for the baseline simulator, SEmulate is applied to a broad range of use cases in Amazon internally with extended capability. In particular, when testing new features in the supply chain systems, building ML models with SEmulate takes a few minutes, whereas integrating the features into a conventional simulation system can take weeks of development effort, or even worse, can be infeasible due to architecture constraints. In such use cases, SEmulate is applied in the same way as illustrated in the synthetic examples, enabling rapid experimentation.

Table 2: Performance of SEmulate against Amazon's retail supply chain simulation. SEmulate reduces the runtime by 90% at a cost of 3% degradation in simulation accuracy. SEmulate enabled 2 additional business use cases which is not feasible with existing simulators.

	relative accuracy gain	relative runtime reduction	#. use case expansion
SEmulate	-3%	-90%	+2

5 CONCLUSION

In this paper, we propose SEmulate, a hybrid framework that integrates ML into simulation. It follows the causal relationship of the systems, where depending on the nature of the sub-component, it decides whether to use emulation or predictive ML models, or simulate according to known mechanism directly. We focus on applying SEmulate for supply chain simulation particularly, where we described the core technology under the SEmulate framework along with illustrative examples to demonstrate its applicability and performance.

We demonstrated how to leverage SEmulate to conduct experimentation on a wide range of examples, both on synthetic data and real-world simulation systems. We showcased the extended capability and improved efficiency of SEmulate on multiple use cases. For example, SEmulate can be used to quickly provide sensitivity analysis on input perturbation to understand how different inputs in the supply chain systems affect the downstream metrics, or the interaction between different systems. It can also be used for counterfactual estimation of system changes or input updates, such as testing the impact of a new demand forecast model, or VLT models.

SEmulate is flexible enough to adjust to different systems' behavior, and obey causal relationship between the system components. It is not limited to the specific applications outlined in this paper, and can be applied beyond supply chain simulation systems. Future work involves generalizing SEmulate to systems where the causal relationship is partially known or unknown, and expanding its capability to not only emulate systems, but also to optimize certain systems.

REFERENCES

Acimovic, J. and V. F. Farias. 2019. "The Fulfillment-Optimization Problem". In *Operations Research & Management Science in the Age of Analytics*, 218–237. INFORMS.

Acimovic, J. and S. C. Graves. 2015. "Making Better Fulfillment Decisions on the Fly in an Online Retail Environment". *Manufacturing & Service Operations Management* 17(1):34–51.

Amazon 2021. "Supply Chain Simulation Environment". https://github.com/amzn/supply-chain-simulation-environment/tree/main.

Amazon 2023. "Amazon.com, Inc. 2023 Annual Report". https://s2.q4cdn.com/299287126/files/doc_financials/2024/ar/Amazon-com-Inc-2023-Annual-Report.pdf. Accessed 1st February 2025.

Farias, V., J. Gijsbrechts, A. I. Khojandi, T. Peng and A. Zheng. 2025, 13-19 Jul. "Speeding up Policy Simulation in Supply Chain RL". In *Proceedings of the 42nd International Conference on Machine Learning*, Volume 267 of *Proceedings of Machine Learning Research*. Vancouver, Canada: PMLR.

Goodfellow, I., Y. Bengio, A. Courville, and Y. Bengio. 2016. *Deep learning*, Volume 1. MIT press Cambridge.

Grow, A. and J. Hilton. 2014. "Statistical emulation". *Wiley StatsRef: Statistics Reference Online*:1–8.

Hadley, G. and T. Whitin. 1963. "Analysis Of Inventory Systems". *Prentice-Hall Inc*.

Jordan, M. I. and T. M. Mitchell. 2015. "Machine Learning: Trends, Perspectives, and Prospects". *Science* 349(6245):255–260.

LeCun, Y., Y. Bengio, and G. Hinton. 2015. "Deep learning". *Nature* 521(7553):436–444.

Liu, J., S. Lin, L. Xin, and Y. Zhang. 2023. "AI vs. Human Buyers: A Study of Alibaba's Inventory Replenishment System". *INFORMS Journal on Applied Analytics* 53(5):372–387.

Lu, X., A. Boukouvalas, and J. Hensman. 2022, 17–23 Jul. "Additive Gaussian Processes Revisited". In *Proceedings of the 39th International Conference on Machine Learning*, Volume 162 of *Proceedings of Machine Learning Research*, 14358–14383. Baltimore, Maryland, USA: PMLR.

Maggiar, A., I. Song, and A. Muharremoglu. 2022. "Multi-echelon Inventory Management for a Non-stationary Capacitated Distribution Network". *SSRN Electronic Journal* https://doi.org/10.2139/ssrn.4154780.

Matloff, N. 2008. "Introduction to Discrete-event Simulation and the SimPy Language". *Davis, CA. Dept of Computer Science. University of California at Davis.*. https://heather.cs.ucdavis.edu/matloff/public_html/SimCourse/PLN/DESimIntro.pdf. Accessed 1st April 2025.

Mitchell, T. M. and T. M. Mitchell. 1997. *Machine Learning*, Volume 1. McGraw-hill New York.

Robinson, S. 2014. *Simulation: the Practice of Model Development and Use*. Bloomsbury Publishing.

Schulz, E., M. Speekenbrink, and A. Krause. 2018. "A Tutorial on Gaussian Process Regression: Modelling, Exploring, and Exploiting Functions". *Journal of mathematical psychology* 85:1–16.

Tucker, S. G. 1965. "Emulation of Large Systems". *Communications of the ACM* 8(12):753–761.

Williams, C. and C. Rasmussen. 1995. "Gaussian Processes for Regression". *Advances in neural information processing systems*.

Young, P. C. and M. Ratto. 2011. "Statistical Emulation of Large Linear Dynamic Models". *Technometrics* 53(1):29–43.

AUTHOR BIOGRAPHIES

XIAOYU LU is an Applied Scientist from Supply Chain Optimization Technology team in Amazon. She received her Ph.D degree in Statistical Science from University of Oxford. Her research interests include machine learning, large language models, Bayesian inference, reinforcement learning and generative models. Her email address is luxiaoyu@amazon.com.

YUJING LIN is a Senior Research Scientist from Supply Chain Optimization Technology team in Amazon. She received her Ph.D degree in Industrial Engineering and Management Science from Northwestern University. Her research interests include simulation input and output uncertainty analysis, meta-modeling, and simulation-based optimization. Her email address is linyujin@amazon.com.

HOIYI NG is a Principal Research Scientist from Supply Chain Optimization Technology team in Amazon. She received her MS degree in Statistics from University of Washington, Seattle. Her research interests include causal inference, graphical causal models, and the intersection between causal inference and large language models. Her email address is nghoiyi@amazon.com.

YUNAN LIU is a Principal Research Scientist in the Supply Chain Optimization Technology team at Amazon. He is also an Adjunct Professor in the ISE Department of NC State University. He earned his Ph.D. in Operations Research from Columbia University. His research interests include stochastic modeling, simulation, optimal control and online learning, with applications to supply chain and call centers. His work was awarded first place in the INFORMS Junior Faculty Interest Group Paper Competition in 2016. His email address is yunanliu@amazon.com. His website is https://yliu48.github.io/.