Outbound Modeling for Inventory Management

Riccardo Savorgnan*
savorgr@amazon.com
Amazon - SCOT Inbound Systems
New York City, NY, USA

Udaya Ghai*
ughai@amazon.com
Amazon AWS - NeuroSymbolic AI
New York City, NY, USA

Carson Eisenach
ceisen@amazon.com
Amazon - SCOT Inbound Systems
New York City, NY, USA

Dean Foster
foster@amazon.com
Amazon - SCOT Inbound Systems
New York City, NY, USA

Abstract

We study the problem of forecasting the number of units drained from each inventory warehouse to meet customer demand, along with the associated outbound shipping costs. The actual drain and shipping costs are determined by complex production systems that manage the planning and execution of customer orders fulfillment. Accurately modeling these processes is critical for regional inventory planning, especially when using Reinforcement Learning (RL) to develop control policies. For the RL usecase, a drain model is incorporated into a simulator to produce long rollouts, which may need to be differentiable. While production systems can be used to recover this transition, they are non-differentiable and too slow and costly to run within an RL training environment. Accordingly, we frame this as a probabilistic forecasting problem, modeling the joint distribution of outbound drain and shipping costs across all warehouses at each time period, conditioned on inventory positions and exogenous customer demand. To ensure robustness in an RL environment, the model must handle out-of-distribution scenarios that arise from off-policy trajectories. We propose a validation scheme that leverages production systems to evaluate the drain model on counterfactual inventory states induced by RL policies. Preliminary results demonstrate the model's accuracy within the in-distribution setting.

ACM Reference Format:
Riccardo Savorgnan, Udaya Ghai, Carson Eisenach, and Dean Foster. 2025. Outbound Modeling for Inventory Management. In *Proceedings of the 1st Workshop on "AI for Supply Chain: Today and Future" @ 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25), August 3, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/XXXXXX.XXXXXX

1 Introduction

Today, reinforcement learning (RL) is used to optimize inventory management systems [7] that must decide order quantities to stock products used to fulfill customer demand. [7] modeled the national buying problem as a periodic review inventory control where at each review period, a single action is taken for each product and the state tracks the total inventory, making the dynamics just accounting. By assuming exogeneity on demand, product level economics, lead-times and other relevant covariates (e.g. holiday features), [7] fit the problem into the framework of an Exogenous Interactive Decision Process (ExoIDP). Unlike worst-case RL, which suffer from exponential sample complexities, backtesting an ExoIDP can be analyzed directly with concentration bounds, and hence reduces to supervised learning. This allows us to reliably backtest on historical data in a gym by replaying the historical exogenous data for each product and tracking the counterfactual trajectories induced by a new policy.

More recently, RL is being proposed to tackle buying and placement supply chain problems which require finer-grained inventory state tracking. In particular, RL has been proposed to solve the the multi-echelon [1] inventory control problem, wherein the action-space is expanded to be an order quantity for each warehouse (sometimes we will refer to these as nodes), or at a warehouse up in the serial line to then be re-routed downstream to warehouses dedicated to demand fulfillment. In such settings, the distribution of inventory across different nodes is necessary for optimal decision making and simulation of the dynamics. While [7] could afford to represent the state with total inventory, in order to properly capture the impact of such policies, we must represent the inventory state of each node[1]. Understanding the evolution of this state after a single period is far more complicated since there are *exponentially* many ways for demand to be fulfilled. Furthermore, each possible configuration of inventory drain yields different fulfillment costs. The way this inventory is fulfilled can be complicated and dependent on many internal systems, along with many other external factors. To maintain the reduction to supervised learning, we would be required to be able to execute production systems with *perfect fidelity*.

While executing production systems may provide a good backtest for a regional inventory control policy, in order to *learn* a policy we need access to a high-speed, accurate, simulator where we evaluate and potentially differentiate through billions of rollouts – a task for which our production systems are too high-latency and costly to run. Instead, we propose a *Drain Model* which will emulate the drain and shipping cost induced by an inventory configuration and incoming arrivals over some specified time period. We consider

*Both authors contributed equally to this research.

[1] Possibly, other coarser granularity might work in some instances.

KDD '25, Toronto, ON, Canada.

ACM ISBN 979-8-4007-1454-2/25/08
https://doi.org/10.1145/XXXXXX.XXXXXX

this as a forecasting problem and train a deep learning model for our task using historic, observational data.

One possible challenge with employing observational data is that it is derived from a distribution induced by historical buying and placement policies. A new regional inventory control policy would produce different state-action trajectories than the historical data distribution. A gym for learning a regional inventory control policy must reliably track the real-world, necessitating off-policy coverage, a common requirement in RL. To illustrate this, consider the following scenarios:

Scenario A (Current Policy): Inventory is placed unevenly, with 10 units in NYC and 0 units in LA. When 4 units of demand arise in LA, fulfillment must occur from NYC. This results in 6 units remaining in NYC, 0 units in LA, and high shipping costs due to cross-regional fulfillment.

Scenario B (Proposed Policy): Inventory is distributed more evenly, with 5 units in both NYC and LA. When the same 4 units of demand arise in LA, they can be fulfilled locally. This results in 5 units remaining in NYC, 1 unit in LA, and low shipping costs due to local fulfillment.

The following table summarizes these two scenarios:

Table 1: Comparison of current and proposed inventory placement policies.

Scenario	Initial Inventory (NYC/LA)	Demand (LA)	Fulfillment (From)	Remaining Inventory (NYC/LA)
A: Current Policy	10 / 0	4	NYC	6 / 0
B: Proposed Policy	5 / 5	4	LA	5 / 1

The critical observation is that historical data only captures outcomes from Scenario A, where inventory is concentrated in NYC. The data distribution lacks examples of how the system would behave under alternative policies like Scenario B, where inventory is spread across warehouses. Consequently, a model trained solely on historical data might fail to predict outcomes for policies that deviate from historical patterns. To address this, our approach involves evaluating the drain model on distributions induced by alternative policies. This requires an oracle capable of generating off-policy validation data. Additionally, fine-tuning the drain model on data produced by the oracle may be necessary to ensure reliable simulation performance.

1.1 Paper structure

2 provides some additional background. 3 presents the mathematical formulation of our problem. 4 provides modeling details. 5 proposes an backtesting oracle for out-of-distribution validation. 6 provides some preliminary empirical results, evaluated in-distribution.

2 Additional Background

Fulfillment Fulfillment decisions are taken by an internal system -for brevity F- that optimizes the fulfillment cost of *sets* of orders, while respecting the original delivery dates that were shown to customers. While it would be ideal to use F as part of a learning framework for inventory control and placement policies, doing so is impractical due to the high latency of the calls and technical limitations when generating counterfactual actions. In this paper we thus set off with the objective of providing an accurate customer order fulfillment emulator, capable of simulating counterfactual fulfillment actions of the system F.

Outbound back-testing oracles We need to validate our system dynamics model on states that are unseen in on-policy data trajectories. The actual system F is the desired instrument for this, as we can call it to generate counterfactuals on an arbitrary inventory state and compare it to the output of the model. To do so, two components are necessary: a) the ability to track per-product inventory along a simulated trajectory and b) customer responses to different products availability. It will thus be possible to validate and collect data for an outbound model with counterfactual state/action trajectories. What we propose in this paper is an algorithm for replaying instances of customers viewing product webpages, which represent potential customer interest in buying an item, simulating customer responses to the counterfactual inventory availability and calling the production algorithm to generate counterfactual inventory drain trajectories. This service could be used to validate an outbound model under "off-policy" state distributions; we discuss this in detail in 5.1.

Customer Demand Correction Delivery dates are influenced by local inventory availability. A customer who is offered a faster delivery has a different probability of buying an item. Furthermore, the delivery date also impacts the ship options available to a customer at checkout, and thus the probability of selecting different options. When changing local inventory availability, we may wish to adjust historical demands and ship options selected by the customer. We utilize a model for correcting demand under different local availability inventory configuration, based on modeling conversions from customer arrivals (i.e. webpage visits or glance views) to an order and ship option as distributed according to a multinomial distribution. We discuss the model in details in F and provide an extension in 5.

3 Mathematical Formulation

3.1 Notation

Throughout the rest of this paper, we will denote matrices by bold uppercase characters. For a set S, we denote the cardinality of that set by $|S|$. The notation $[N]$ denotes the sequence of natural numbers through N (i.e. $\{1, ..., N\}$). For a matrix $\mathbf{M}$, we denote the element in the i-th row and j-th column of the matrix as $\mathbf{M}_{i,j}$. We denote by $\mathbb{R}_{\geq 0}$ the nonnegative reals and $\mathbb{R}_{+}$ the positive reals. Similarly we denote $\mathbb{Z}_{\geq 0}$ and $\mathbb{Z}_{+}$ for the nonnegative and positive integers, respectively. Denote by $(\cdot)^{+}$ the positive part operator. We use $\Delta^{d} \subseteq \mathbb{R}_{\geq 0}^{d+1}$ to represent the d simplex.

3.2 Outbound process

In this section we describe the process of how outbound is determined, starting from a customer's interest in a product. Denote by $\mathcal{A}$ the set of products managed by the retailer, $\mathcal{F}$ the set of warehouses from which the retailer can fulfill orders and $\mathcal{Z}$ the set of customer regions (for example a region can be defined by grouping all addresses sharing the first 2 digits of the zip code, also called Zip2s).

First, the customer arrives at the detail page -i.e. the webpage- for a product $i \in \mathcal{A}$. They are then shown a *promise* p of how quickly the product can be delivered to them. There are multiple possible promises – including one for *product is unavailable* – and we denote the set of all promises by $\mathcal{S}$. After viewing the promise, the customer decides whether or not to purchase the product. When purchasing the product, the customer selects a shipping speed option (for example, next-day or two-day) o that is no faster than the original promise shown. The set of all ship speeds is denoted by $\mathcal{O}$, and by convention we always include a ship-speed option that corresponds to *no order* being placed. Once the order is placed, the fulfillment policy F determines which warehouse to fulfill from.

Exogenous and control processes. Having described how the outbound process works, we define the processes that we wish to model and which determine the transition dynamics. We index time series processes by $t \in \mathbb{Z}_{\geq 0}$ and all time series are discretized to the same granularity (e.g. weekly). The time step t corresponds to the time interval $[t, t+1)$.

Glance Views A *glance view* consists of the associated product, the region of the customer, a promise shown, a ship option (SO) selected at checkout, and a quantity in $\mathbb{R}_{\geq 0}$ ordered. Formally, a glance view v belongs to $\mathcal{V} := \mathcal{A} \times \mathcal{Z} \times \mathcal{S} \times \mathcal{O} \times \mathbb{R}_{\geq 0}$. We will denote by H_Z the mapping from v to its region $z \in \mathcal{Z}$, and H_A, H_S, H_O analogously, with $H_Q(v)$ producing the order quantity. At each time period t, and for each product i, there is a sequence of glance views $G_t^i = (v_{t,\tau}^i)_{\tau \in \mathbb{Z}_+}$,where each glance view is associated with item i, which are indexed in increasing order of time τ that customers arrive at the detail page (each individual glance view typically corresponds to a different costumer). Next, we denote the total regional glance views for product i at time t as $g_t^i \in \mathbb{R}^{|\mathcal{Z}|}$ where $g_t^i := (g_t^{i,1}, \ldots, g_t^{i,|\mathcal{Z}|})$ and

$$g_t^{i,z} := |\{v \in G_t^i : H_Z(v) = z\}|.$$

Active warehouses The set of active warehouses changes over time, as new ones are built. We denote with $\tilde{\mathcal{F}}_t \subseteq \mathcal{F}$ the set of warehouses that are active at time t.

Inventory We denote the inventory at a warehouse f of product i at the end of period t as $I_t^{i,f} \in \mathbb{R}_{\geq 0}$ and the vector of inventory for a product i at time t as $I_t^i \in \mathbb{R}_{\geq 0}^{|\mathcal{F}|}$ where $I_t^i := (I_t^{i,1}, \ldots, I_t^{i,|\mathcal{F}|})$.

Outbound units We denote the units outbounded of product i from a warehouse f during time t as $o_t^{i,f} \in \mathbb{R}_{\geq 0}$. The vector of outbound for a product i at time t is defined as $o_t^i \in \mathbb{R}_{\geq 0}^{|\mathcal{F}|}$ where $o_t^i := (o_t^{i,1}, \ldots, o_t^{i,|\mathcal{F}|})$.

Shipping cost We denote the total shipping cost from a warehouse f for product i at time t as $c_t^{i,f}$, and the vector of shipping costs for item i at time t as $c_t^i \in \mathbb{R}_{\geq 0}^{|\mathcal{F}|}$ where $c_t^i := (c_t^{i,1}, \ldots, c_t^{i,|\mathcal{F}|})$. Note that the warehouse-level costs are the sum of the unit level costs for all units outbounded from that warehouse. We discuss several ways to perform this unit-level accounting in D.

Stowed units We denote the units stowed (i.e. received and available for customer fulfillment) of product i from a warehouse f during time t as $a_t^{i,f} \in \mathbb{R}_{\geq 0}$. The vector of stowed units for a product i at time t is defined as $a_t^i \in \mathbb{R}_{\geq 0}^{|\mathcal{F}|}$ where $a_t^i := (a_t^{i,1}, \ldots, a_t^{i,|\mathcal{F}|})$.

Historical covariates Additionally, we may have other historical covariates of interest (such as holiday indicators, etc) that we may wish to include as part of the outbound modeling. We denote these as y_t^i.

3.3 Formulation as a Forecasting Problem

Outbound quantities and shipping costs depend on glanceviews (customer arrivals) and inventory position, but the arrivals and conversions are stochastic. We thus cast this as a probabilistic forecasting task, seeking

$$p(o_t^i, c_t^i \mid H_t^i, \theta) \;=\; p(o_t^i \mid H_t^i, \theta)\; p(c_t^i \mid o_t^i, H_t^i, \theta),$$

where θ are learnable model parameters.

4 Proposed ML-Based Forecasting Approach

4.1 Outbound Distribution

Similarly to [10], because many product-warehouse-time combinations have small or zero outbound, we allocate explicit probability mass to those values, while accommodating unbounded large outbounds via the quantile-based tail. A fully discrete model would require enormous support, while a pure quantile-based approach often struggles with calibration when much of the data is zero or near zero. Consequently, we introduce a hybrid *discrete-plus-quantile* model for $p(o_t^i \mid H_t^i, \theta)$. Specifically, we define discrete probabilities $p_{\mathrm{disc},k}(H_t^i, \theta)$ for integer values $k = 0, 1, \ldots, n_d - 1$, with the remaining probability mass collected in

$$p_{\mathrm{disc},n_d}(H_t^i, \theta) \;:=\; p\big(o_t^i \geq n_d \mid H_t^i, \theta\big).$$

A learned quantile-based CDF $F_q(k) := F_{\mathrm{quant}}(k \mid H_t^i, \theta)$ then refines how this tail mass is distributed. Formally,

$$p(o_t^i = k \mid H_t^i, \theta) = \begin{cases} p_{d,k}(H_t^i, \theta), & k < n_d, \\ p_{d,n_d}(H_t^i, \theta)\,\Big[F_q(k) - F_q(k-1)\Big], & k \geq n_d. \end{cases} \tag{1}$$

4.2 Cost Distribution.

Shipping costs do not exhibit the same degree of sparsity due to the conditioning on outbound quantities, so we adopt standard quantile forecasting methods [2, 11], predicting n_q quantiles of c_t^i conditional on the realized outbound o_t^i. Several techniques exist for interpolating these quantiles and drawing samples [8].

4.3 Loss function

The model is trained on a linear combination of 5 losses. First, for the discrete outbound prediction, we use cross entropy loss. Then for the quantile predictions of the model, both in cost and in outbound, we use a combination of quantile loss and negative log-likelihood (NLL) from the quantile-interpolated distribution. We emphasize NLL here because likelihood is a *proper-scoring rule* [4] for sequence generation, while quantile-loss is only proper for a single prediction. Given the intent to use these models for RL simulators, this distinction is relevant.

4.4 Network architecture

We follow the MQ-Forecaster framework [2, 11], with customized design to account for sharing of information between different warehouses for aligning warehouse-level data with location-level data. The architecture starts by using a wavenet encoder on two parallel streams of data (glanceview time-series data with location-granularity and warehouse time-series data). This step acts on each node/location locally. The next component involves sharing information between node-level embeddings and location-level embedding via either a bidrectional RNN or a Transformer. These are chosen such that the model can flexibly adapt to new warehouses. In [5], [6] the authors utilize graph networks to match supply and demand at different granularitieis. Similarly, [12] utilizes the cross-attention mechanism to let demand streams attend to each other, resulting in information exchange. In the same spirit, we utilize a cross-attention layer to join the two embedding streams from nodes and locations, allowing for the supply-demand matching we aim to capture. Finally, these embeddings are decoded via MLPs to representations (quantiles and logits) in order to sample outbound. The Outbound is provided as an input for a decoder for a cost head. In training, we use teacher forcing, providing the cost decoder the actual outbound rather than a sample from the model. See Figure 1 for a network schematic.

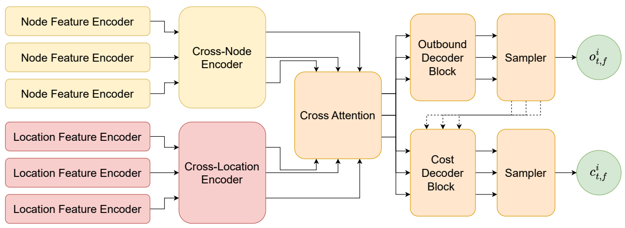

Figure 1: Architecture schema of our proposed model. Dashed Lines between the sampler and Cost Decoder Block represent teacher-forcing during training.

5 An off-policy backtesting oracle

We describe a methodology to implement an oracle for backtesting out-of-distribution trajectories. The high level idea is to replay each historical glance view, get its promise from a production system, convert it to an order, conditionally on this promise, and then pass this to system F. With the counterfactual fulfillment plan from F, we can estimate a shipping cost with a model. This whole process will be run sequentially through time – *not parallelized by product* – in order for the backtest to capture the effect of the counterfactual inventory placement on “multi-shipments”, i.e. shipments where two or more items are packed together and their shipment cost is tied to the cost of shipping the box rather than the individual items –we give more details in appendix D–. We now define it precisely.

5.1 Conditional Glance View Conversions

In our backtesting methodology, accurately modeling how historical glance views convert into orders under new promises is crucial. To achieve this, we introduce a *conditional* conversion model that leverages historical data to inform counterfactual scenarios. This can be viewed as a speed-aware extension of availability correction used in [7]. This section elucidates the conditional glance view conversion processs.

Motivation Independent sampling of glance view conversions can lead to high variance and unrealistic scenarios, especially when multiple related orders influence each other over time. By conditioning on historical outcomes, we ensure that the simulated conversions remain consistent with observed behaviors, which may potentially produce more reliable backtest results.

Conceptual Framework Consider a glance view v that historically resulted in a specific ship option under a given promise. When evaluating a new promise p, we aim to determine the probability of each possible ship option while respecting the historical decision-making process. This is achieved by conditioning on the historical promise $H_S(v)$ and the historical outcome $H_O(v)$, effectively reusing the underlying randomness that led to the original conversion.

Cumulative Distribution Function (CDF) For each pproduct i at time t and promise p, we define the cumulative conversion rate over ship options O as:

$$\bar{R}_t^{i,p,o} = \sum_{o' \leq o} \hat{R}_t^{i,p,o'}$$

where $\hat{R}_t^{i,p,o}$ is the estimated probability that a glance view for product i at time t under promise p converts to ship option o. $\bar{R}_t^{i,p,o}$ denotes the probability of conversion at speed o or faster. As conversion rates increase with promise, $\bar{R}_t^{i,p,o}$ can be assumed to be monotone increasing as promise speed increases.

This cumulative conversion rate allows us to interpret the conversion process as follows: Imagine that the historical order outcome was generated by drawing a uniform random variable $U \in [0, 1]$ and selecting ship option o if

$$\bar{R}_t^{i,p,o-1} < U \leq \bar{R}_t^{i,p,o}$$

where $\bar{R}_t^{i,p,o-1}$ is the cumulative conversion rate just faster than ship option o.

Conditioning on Historical Outcomes Given a historical glance view with promise $H_S(v)$ and outcome $H_O(v)$, we can infer that the underlying U must have fallen within a specific interval that led to the observed outcome. Specifically, if $H_O(v) = o_{\text{hist}}$, then:

$$\bar{R}_t^{i,H_S(v),o_{\text{hist}}-1} < U \leq \bar{R}_t^{i,H_S(v),o_{\text{hist}}}$$

By conditioning on this interval, we restrict U to lie within $\left(\bar{R}_t^{i,H_S(v),o_{\text{hist}}-1}, \bar{R}_t^{i,H_S(v),o_{\text{hist}}}\right]$. This ensures that when evaluating a new promise p, the same U is used, maintaining consistency in the conversion process. Because of the ordering, an increase in the promise can only increase the probability of converting at faster speeds and so if promise increases (or stays the same) and an order converted historically, this sampling process will result in the order converting again, though potentially at a faster ship option.

Table 2: Validation quantile loss, nll, along with discrete cross-entropy (ce). L is for the final combined validation loss. Closest-node Baseline(Appendix E) represents our modeling of a greedy closest-node baseline. RNN and Transformer represent using transformer blocks or a Bidirectional RNN for the Cross-X-encoders in Figure 1, while sales indicates whether a feature for sales is included. These are shaded as gray since sales cannot be used in an RL dynamics model, as they are endogenous.

	Cost				Outbound					L
	q10	q50	q90	nll	q10	q50	q90	nll	ce	
Closest node	-	-	-	-	0.614	0.671	0.700	15.1	8.15	-
RNN	0.052	0.147	0.086	1.442	0.050	0.144	0.081	2.638	0.754	4.394
Transformer	0.052	0.148	0.087	1.442	0.050	0.145	0.082	2.634	0.758	4.355
RNN w sales	0.052	0.145	0.084	1.429	0.052	0.150	0.082	2.653	0.759	4.274
Transformer w sales	0.052	0.144	0.083	1.412	0.049	0.142	0.078	2.621	0.752	4.247

Table 3: Samples are taken from the model and aggregated at a national and regional level. Multiplicative calibration is done for empirical quantiles and mean of outbound resulting in calibration slopes (p10, p50, p90), OLS slope, and CRPS for each model at Total and Regional granularity.

Model	Aggregation	p10 slope	p50 slope	p90 slope	OLS slope	CRPS
Closest node	Total	0.75	0.94	1.11	0.91	48.51
RNN	Total	0.90	0.98	1.04	1.07	34.85
Transformer	Total	0.94	0.94	0.93	0.94	20.60
Closest node	Regional	0.13	0.42	1.30	0.28	26.90
RNN	Regional	0.91	0.98	1.00	0.96	6.25
Transformer	Regional	0.96	0.96	0.96	0.96	4.11

Conditional Probability Calculation Given the constrained U from the historical outcome, we compute the conditional probability of each ship option o under a new promise p. The conditional probability is defined as:

$$\Pr[o \mid p, H_S(v), H_O(v)] = \left(\frac{\min\left(\bar{R}_t^{i,p,o}, \bar{R}_t^{i,H_S(v),H_O(v)}\right) - \max\left(\bar{R}_t^{i,p,o-1}, \bar{R}_t^{i,H_S(v),H_O(v)-1}\right)}{\hat{R}_t^{i,H_S(v),H_O(v)}} \right)^+$$

For an illustrative example of the conditional glanceview conversion process, see Appendix B. This glanceview conversion is integrated into an outbound oracle that loops through glanviews, converts them, calls FTP and tracks inventory which is detailed in Appendix C.

6 Empirical Results

We train our models on 50k sampled products between July 2022 and December 2023, validating on the same product set between January 2024 and June 2024. Models are trained using a linear combination quantile loss, negative log-likelihood and cross-entropy for the discrete outbound prediction, as discussed in Section 4.3. We train a model that used a bidirectional RNN cross-node and cross-location encoder and consider a replacement which replaces this with two transformer blocks, finding fairly negligible difference. To study the impact of glanceview conversion to the model, we use a model that has access to sales instead of just glanceviews, though this is not available for an RL dynamics model. These models do not demonstrate significant improvements in outbound prediction, but do produce improvements in the cost prediction. This could make sense as the finer grained location of demand could provide signal for shipping cost. We perform two primary types of analysis for models. First, we evaluate on the metrics used training, as can be seen in Table 2. Next, for 5k products per model we draw 128 samples from the predicted distributions and perform aggregations at national and regional levels. We observe multiplicative calibration of the empirical quantiles and mean[2] along with Continuous Ranked Probability Score (CRPS) in Table 3. Finally, we demonstrate that our model is relatively well calibrated over time in both quantiles and in it’s discrete predictions in Figure 2.

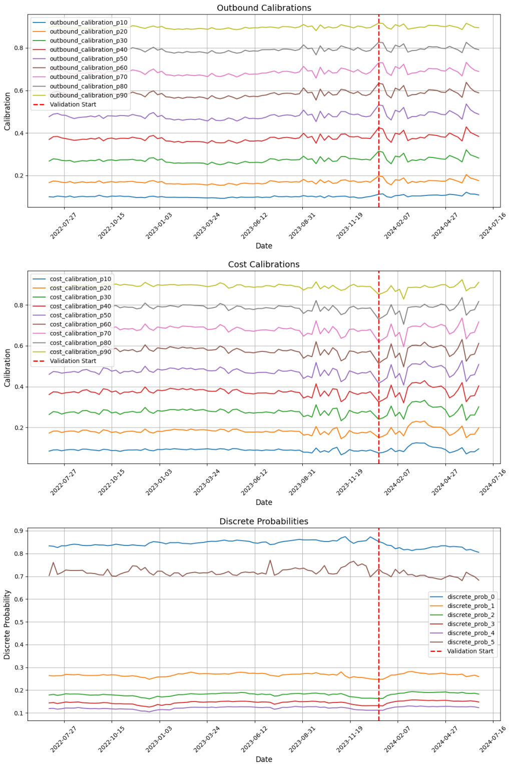

Figure 2: Calibrations for quantiles and the discrete predictions. Discrete calibrations represent probability of k given true outbound is k. Final probability 5 represents probability >= 5.

[2] We regress the ground truth using the empirical quantiles and mean as feature, where calibration means 1

References

[1] A. J. Clark and H. Scarf. 1960. Optimal policies for a multi-echelon inventory problem. *Management Science* 6, 4 (1960), 475–490. doi:10.1287/mnsc.6.4.475

[2] Carson Eisenach, Yagna Patel, and Dhruv Madeka. 2020. MQTransformer: Multi-Horizon Forecasts with Context Dependent and Feedback-Aware Attention. arXiv:2009.14799

[3] Marco Geraci and Alessio Farcomeni. 2021. Mid-quantile regression for discrete responses. arXiv:1907.01945 [stat.ME] https://arxiv.org/abs/1907.01945

[4] Tilmann Gneiting and Adrian E. Raftery. 2007. Strictly Proper Scoring Rules, Prediction, and Estimation. *J. Amer. Statist. Assoc.* 102, 477 (2007), 359–378. doi:10.1198/016214506000001437

[5] Hyung il Ahn, Young Chol Song, Santiago Olivar, Hershel Mehta, and Naveen Tewari. 2024. GNN-based Probabilistic Supply and Inventory Predictions in Supply Chain Networks. arXiv:2404.07523 [cs.AI] https://arxiv.org/abs/2404.07523

[6] Mengjin Liu, Yuxin Zuo, Yang Luo, Daiqiang Wu, Peng Zhen, Jiecheng Guo, and Xiaofeng Gao. 2024. Weather-Conditioned Multi-graph Network for Ride-Hailing Demand Forecasting. In *Service-Oriented Computing: 22nd International Conference, ICSOC 2024, Tunis, Tunisia, December 3–6, 2024, Proceedings, Part II* (Tunis, Tunisia). Springer-Verlag, Berlin, Heidelberg, 341–356. doi:10.1007/978-981-96-0808-9_26

[7] Dhruv Madeka, Kari Torkkola, Carson Eisenach, Anna Luo, and Dean Foster. 2022. Deep Inventory Management. arXiv:2210.03137

[8] Vincent Quenneville-Belair, Malcolm Wolff, Brady Willhelme, Dhruv Madeka, and Dean Foster. 2023. Distribution-free multi-horizon forecasting and vending system. In *KDD 2023 International Workshop on Mining and Learning from Time Series (MileTS)*. https://www.amazon.science/publications/distribution-free-multi-horizon-forecasting-and-vending-system

[9] Bruno Santos and Heleno Bolfarine. 2015. Bayesian quantile regression analysis for continuous data with a discrete component at zero. arXiv:1511.05925 [stat.ME] https://arxiv.org/abs/1511.05925

[10] Zirui Wang and Tianying Wang. 2027. A Semiparametric Quantile Single-Index Model for Zero-Inflated and Overdispersed Outcomes. *Statistica Sinica* (2027). doi:10.5705/ss.202024.0104

[11] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. 2017. A multi-horizon quantile recurrent forecaster. In *NIPS Time Series Workshop.*

[12] Yiling Wu, Yingping Zhao, Xinfeng Zhang, and Yaowei Wang. 2024. Spatial–Temporal Correlation Learning for Traffic Demand Prediction. *IEEE Transactions on Intelligent Transportation Systems* PP (11 2024), 1–14. doi:10.1109/TITS.2024.3443341

A Model Details

A.1 Features used

Table 4: Node Features

Feature Name	Description
`outbound`	Fulfilled demand from this node.
`shipping_cost`	Cost or shipping metric for this node.
`node_location`	GPS coordinates for this node.
`available_inv`	Current available inventory at node.
`is_warehouse_active`	Binary indicator if node is active.

Table 5: Location Features

Feature Name	Description
`total_gv`	Aggregate glanceviews (webpage visits) for this location.
`zip_location`	GPS coordinates for this location.

Table 6: Holiday Features

Feature Name	Description
`distance_to_event_*`	distance measure in days to yearly seasonal events.

A.2 Loss and optimization details

The loss is configured as a fixed linear combination of cost nll, cost quantile loss, cross entropy on the discrete outbound predictions, nll on the outbound for the quantile part of the distribution and quantile loss for the outbound. The weights are $[0, 4, 2, 0.3, 6]$, which were chose to approximately normalize each loss to scale close to 1 during training, though this is quite arbitrary. Understanding how these losses impact a downstream metric is an interesting area for improvement.

Models are trained for $200 - 300$ epochs or until validation loss seems steadily increasing, which occured with transformer models. Losses typically converged much quicker, reaching near final loss within 20 epochs.

Models were trained with DDP on 8 40 GB A100 GPUs, allowing for batch sizes on the order of 160 for the models using transformer encoders and 256 for models using the RNN encodes which required less memory.

A.3 Model hyperparameters

Output Tensors:

- Outbound distribution: 6 discrete logits plus 9 quantiles.
- Cost distribution: continuous quantiles.

Key Hyperparameters:

- `hidden_size` = 64, `attention_heads` = 8.
- `rnn_layers` = 2, or `transformer_layers` = 2
- `mlp_depth` = 3, `dropout` = 0.1.
- `atrous_rates` = [1, 2, 4] (dilations in CNN).
- `quantiles` = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9].
- `num_outbound_logits` = 6 (6 discrete demand buckets so outbound >= 5 is predicted using quantiles).

A.4 Architecture Outline

(1) Temporal CNN Encoders Both `node_t` and `location_t` are first concatenated (per time step) with the `dist_t` features, then each passes through a stack of **3 dilated convolutions** (WaveNet-style).

- Each convolution has: kernel_size = 2, out_channels = `hidden_size` = 64, dilation $\in \{1, 2, 4\}$.
- Each layer is followed by `ELU` and a padding step to preserve sequence length.

This produces node embeddings $x \in \mathbb{R}^{B \times T \times N \times 64}$ and location embeddings $z \in \mathbb{R}^{B \times T \times L \times 64}$.

(2) Channel RNN Encoding Since `transformer_layers` = 0, we use a bidirectional RNN on the channel dimension (N or L):

- RNN input_size = 64, hidden_size = 64/2 = 32 per direction, num_layers = 2.
- The node embeddings ($B \times T \times N \times 64$) are reshaped to $(B \times T) \cdot N$ mini-batches, passed through the RNN, then reshaped back. Location embeddings similarly.

We obtain updated node embeddings x_r and location embeddings z_r, both of shape $(B \times T \times \{N \text{or} L\} \times 64)$.

(3) Cross-Attention (Nodes ⟵ Locations) A multi-head cross-attention module (`attention_heads` = 8) allows each node to attend over location embeddings:

$$\mathrm{Att}(x_r, z_r) \in \mathbb{R}^{B \times T \times N \times 64}.$$

This becomes the "outbound head."

(4) Outbound MLP Decoder A depth-3 MLP (with `in_features` = 64, `out_features` = `num_outbound_logits` + Q = 10 + 9 = 19) produces:

$$\underbrace{o\_\text{logits}}_{(10 \text{ channels})}, \quad \underbrace{o\_\text{quantiles}}_{(9 \text{ channels})}.$$

The logits model discrete demand buckets; the quantile channels (via softplus + cumsum) ensure monotonic quantile outputs.

(5) Cost Quantile Transformer We map (x_r, z_r) to cost quantiles similarly:

- Optionally concatenate the outbound draw (if available) to x_r.
- A second cross-attention with location embeddings.
- A depth-3 MLP outputs cost quantiles, shape $(B \times T \times N \times 9)$.

Summary Using the above 5-step pipeline, our model creates distributional predictions of both outbound demand (discrete + quantile) and cost (quantile). The default dimensioning follows:

- **CNN Layers:** 3 dilated convs (k = 2, dilation = 1,2,4), each output size = 64.
- **RNN:** Bidirectional, 2 layers, hidden size = 64 total (32 per direction).
- **Attention:** 8 heads.
- **MLPs:** 3-layer fully connected, from 64 up to final channel count (19 for outbound, 9 for cost).

B Customer Interest Conversion Example

To concretize the conditional conversion mechanism, consider the following example involving a single product and a single time period.

- **Ship Option Cateogries**: $\mathcal{O} = \{1\text{d}, 2\text{d}, 3\text{d+}, \text{NoOrder}\}$, indexed as $o \in \{0, 1, 2, 3\}$.
- **Promise Categories**: $\mathcal{S} = \{1\text{d}, 2\text{d}, 3\text{d+}, \text{Out-of-Stock}\}$,.
- **Historical Data** for product i at time t under historical promise $p_h = H_S(v)$:

Table 7: Historical Conversion Probabilities for Product i at Time t under Promise p_h = 2d

Ship Option	Probability	CDF	U Range
1d	0.00	0.00	[0.00, 0.00)
2d	0.20	0.20	[0.00, 0.20)
3d+	0.10	0.30	[0.20, 0.30)
NoOrder	0.70	1.00	[0.30, 1.00]

Suppose the historical outcome was $H_O(v)$ = 2d, which implies that the underlying U fell within the range $[0.0, 0.2)$.

Backtest Scenario:

- **New Promise**: p_b = 1d (improved promise).
- **Conversion Probabilities under** p_b:

Table 8: Conversion Probabilities for Product i at Time t under New Promise p_b

Ship Option	Probability	CDF	U Range
1d	0.15	0.15	[0.00, 0.15)
2d	0.15	0.30	[0.15, 0.30)
3d+	0.10	0.40	[0.30, 0.40)
NoOrder	0.40	1.00	[0.40, 1.00]

Conditional Probability Calculation:

Given that the historical $U \in [0.0, 0.2)$, we examine how this interval overlaps with the new CDF under p_b to determine the conditional probabilities for each ship option.

Table 9: Overlap of Historical U with New Promise p_b CDF

Ship Option	New CDF Range	Overlap with $[0.0, 0.2)$	Overlap Length	Conditional Probability
1d	[0.00, 0.15)	[0.00, 0.15)	0.15	$\frac{0.15}{0.20} = 0.75$
2d	[0.30, 0.40)	[0.15, 0.20)	0.05	$\frac{0.05}{0.20} = 0.25$
3d+	[0.40, 0.50)	None	0.00	0.00
NoOrder	[0.50, 1.00]	None	0.00	0.00

Summary of Conditional Conversion:

Table 10: Conditional Conversion Probabilities under New Promise p_b

Ship Option	Conditional Probability
1d	0.75
2d	0.25
3d+	0.00
NoOrder	0.00

Interpretation of Results From Tables 9 and 10, we derive the following conditional probabilities:

- **NoOrder**: The overlap is zero, resulting in a conditional probability of 0.00. This means that, under the new promise p_b, a historical U that previously led to an order cannot result in a "NoOrder" outcome under the faster promise.
- **1d**: The overlap length of 0.15 divided by the original interval length of 0.20 yields a conditional probability of 75%. Therefore, there is an 75% chance that the ship option becomes "1d" under the new promise.
- **2d**: The overlap length of 0.05 divided by the original interval length of 0.20 yields a conditional probability of approximately 25%. Thus, there is a 25% chance that the ship option remains "2d" under the new promise.
- **3d+**: The overlap is zero, resulting in a conditional probability of 0.00. This indicates that, under the new promise p_b, there is no probability of selecting the "3d+" ship option based on the historical U.

This conditional conversion ensures that the historical relationship between promises and outcomes is maintained while adapting to new promises, thereby producing realistic and coherent order outcomes in the backtest scenario.

C Oracle

C.1 Outbound estimation - production systems

We will estimate outbound need to define the production systems we need to call. The Promise system can be viewed as a function that maps an inventory configuration for an product to a promise for all $|\mathcal{Z}|$ regions of the form $F_{pr} : \mathbb{R}^{|\mathcal{F}|}_{\geq 0} \times \mathcal{A} \to \mathcal{S}^{|\mathcal{Z}|}$. The second system that we need to call to implement an outbound oracle is F, which can be viewed as a stateful system that ingests an inventory configuration $I \in \mathbb{R}^{|\mathcal{F}|}_{\geq 0}$, customer region $z \in \mathcal{Z}$ and order and eventually produces a warehouse for the order to be shipped from.

C.2 Ship cost estimation

Neither of the two production systems described above will give us a ship cost estimate (the other component of the oracle). For our purposes, we require the ability to sample from the shipment-level distributions. We have an estimated distribution for each time, product, node and region combination, and we denote these as $\hat{P}^{i,f,z}_t$. See D.1 for more detail.

C.3 Putting it all together

1 describes an oracle that takes an inventory state and sequence of glance views and returns a vector of outbound quantities for each product. We use Python-style array indexing for ease of exposition. Observe that this oracle requires a number of service calls linear in the number of orders (not glance views). Note that final accounting of an order is delayed from when it is placed, as is done in practice. This is purposeful to accurately emulate multi-shipments, though the control flow will be more complicated in real implementation.

Algorithm 1 Outbound Oracle

```
Input: $I^i \in \mathbb{R}^{|\mathcal{F}|}_{\geq 0}$, $\mathcal{G} = (v_s)_{s \in \mathbb{Z}_+}$, $\{\hat{R}^{i,p,o}\}$, $\{\hat{P}^{f,z}\}$
  $O^i \leftarrow (0, \ldots, 0) \forall i$ // Total outbound units per node
  $C^i \leftarrow (0, \ldots, 0) \forall i$ // Total ship cost per node
  $P \leftarrow F_{pr}(I)$
  for $v \in \mathcal{G}$ do
    $z' \leftarrow H_Z(v)$ // Get region for current glance view
    $i' \leftarrow H_A(v)$ // Get product for current glance view
    $p' \leftarrow P[z']$ // Get promise for current glance view
    $o' \sim \Pr[\cdot | p', H_S(v), H_O(v)]$ // Sample conversion to order
    if $o'$ != NoOrder then
      // Order is placed, F adds to fulfillment set
      $(I^{i'}, z', o') \rightarrow$ F //Send F orders
    end if
    if F.updated() then // F assigns previous orders
      $f'', i'', o'' \leftarrow$ F // Get region for current glance view
      $c' \sim \hat{P}^{i'', f'', z''}$ // Sample ship cost
      $O^{i''}[f'] \leftarrow O^{i''}[f''] + 1$
      $I^{i''}[f'] \leftarrow O^{i''}[f''] - 1$
      $C^{i''}[f'] \leftarrow C^{i''}[f''] + c'$
      $P \leftarrow F_{pr}(I, i)$
    end if
  end for
Output: $O^i$, $C^i$ for all products $i$
```

D Shipping costs

Here we address two important facts regarding the cost of fulfillment: its definition and the implications of such definition for the *evaluation* of an inventory placement policy. In general, the fulfillment cost can be decomposed as a sum of the costs incurred for packing and delivering a box to a customer. A desirable decomposition method is one that it is representative of the total costs incurred at a company level, but also causally attributes certain costs at a box-unit level. The guarantee on the total costs allows our simulated evaluation to be consistent with what we would see in a real environment, while the box-unit level attribution allow us to attribute a granular reward based on the outcome of inventory placement actions -i.e. the resulting fulfillment trajectory- that can be used to learn an inventory placement policy. An example of this attribution is to start from a "ground truth" cost of shipping a box of a given weight and size, and split the shipment cost of the box's final delivery to the customer across units within that box, according to some criteria -i.e. their volume and weight-.

The latter example highlights the implication of such attribution method: shipment costs may not be independent across ordered products, as for example in this case packing two items together significantly modifies the fulfillment cost of both. Changing the shipment modality thus implies the counterfactual cost **has** to be computed conditionally to all shipped units that *could have* been packed together, across all products. So while our simulator operates at a product level, any ground-truth backtesting methodology **must** account for counterfactual cross-product effects to correctly *evaluate* a policy.

D.1 Estimating the Shipping Cost

We here provide a way to estimate the shipping cost of a box of items, based on some generic characteristics of a shipment. This estimate allows us to recreate a cost within the simulator described in C. In fact, for the purpose of *evaluating* the results of an inventory control policy in a simulator, we obviously desire that the simulated costs are representative of the ones that we'd incur in reality.

Since boxes have clearly attributed costs of shipments, by design, the cost accounting represents the total cost incurred for shipments. As such, if we can recreate an unbiased estimate in our simulator, then the cost that our simulator will output should also be representative of the total bill incurred. Naturally, the real costs depend on characteristics of the shipment (characteristics of the package, distance traveled and so on), which are influenced by the inventory control process. We can make our forecast depends on such characteristics, which allows us to forecast costs accurately even when off-policy.

As an exercise, we regress (OLS) the shipping cost of a box $sc^{\mathcal{B}_k}$ over a set of features that reasonably depend from the inventory control process and show the results in Table 11. For proprietary reasons, we mask features and all the regression coefficients. All of them are statistically significant with $p < 0.001$, and the high R-squared is indicative of a fairly well performing model, regardless of its simplicity.

Table 11: OLS regression of the "ground-truth" shipping cost per box $sc^{\mathcal{B}_k}$ in respect of physical characteristics of the shipment.

Dep. Variable:	$sc^{\mathcal{B}_k}$	**R-squared:**	0.509
Model:	OLS	**Adj. R-squared:**	0.509
Method:	Least Squares	**F-statistic:**	3993.
No. Observations:	50000	**Prob (F-statistic):**	0.00
Df Residuals:	49986	**Log-Likelihood:**	-97744.
Df Model:	13	**AIC:**	1.955e+05
Covariance Type:	nonrobust	**BIC:**	1.956e+05

E Baseline Model: Closest-node

In this section we discuss our choice of Closest-node as a baseline, present its detailed results and discuss its performance. Closest-node describes an algorithm that, given a sequence of customer orders, sequentially assigns the closest node with inventory to fulfill the next order, until all orders have been fulfilled or inventory has ran out. Closest-node is anecdotally accepted to be a good heuristic as well as being human interpretable.

Algorithm: We here describe our implementation of this algorithm, specifically how it generates a vector (over warehouses) of outbound quantities for a given product and time. For every product and time interval, we call for the regional vectors of glance views $g^i \in \mathbb{Z}^{|\mathcal{Z}|}$, national conversion rate $c^i \in [0, 1]$, inventory in each warehouse $I^i \in \mathbb{Z}_{\geq 0}^{|\mathcal{F}|}$ and a distance function between regions and warehouses $d : \mathcal{Z} \times \mathcal{F} \longrightarrow \mathbb{R}_{\geq 0}$.

Algorithm 2 Closest-node

```
Input: $I^i \in \mathbb{Z}_{\geq 0}^{|\mathcal{F}|}, g^i \in \mathbb{Z}_{\geq 0}^{|\mathcal{Z}|}, c^i \in [0, 1], d$
  $O^i \longleftarrow 0$  // Initiate outbound vector at 0
  for $z \in \mathcal{Z}$ do
    $s_z^i \sim \text{Binomial}(\theta = c^i, n = g_z^i)$  // Sample conversion to order via Binomials
  end for
  while $\sum_z s_z^i > 0 \wedge \sum_f I_f^i > 0$ do
    $p^i \longleftarrow s^i / \sum_z s_z^i$  // Get share of demand per region
    $z' \sim \text{Multinomial}(\vec{\theta} = p^i, n = 1)$  // Sample region from Multinomial
    $\mathcal{F}' \longleftarrow \{f \in \mathcal{F} : I_f^i > 0\}$  // Find set of warehouses with inventory
    for $f \in \mathcal{F}'$ do
      $\delta_f^i \longleftarrow d(z', f)$ // Get vector of distances
    end for
    $j' \sim \text{Multinomial}(\vec{\theta} = \text{softmax}(-\delta^i), n = 1)$  // Sample closest warehouse
    $O_{j'}^i \longleftarrow O_{j'}^i + 1$  // Update outbound
    $I_{j'}^i \longleftarrow I_{j'}^i - 1$  // Update inventory
    $s_{z'}^i \longleftarrow s_{z'}^i - 1$  // Update orders
  end while
Output: $O^i$
```

Products, times and samples are considered independent of each other; in practice we thus vectorized the algorithm across these dimensions with the appropriate considerations. Note also that this algorithm relaxes the notion of closest node with a softmax sampling, which allows inventory to be drawn, with probability, from other nodes in the proximity. This is general enough to represent the Closest-node notion: as we tune the multiplier hyperparameter, the softmax sampling acts more and more deterministically, degenerating in an argmin function and thus coinciding with the closest node notion.

Experiment and Results: We ran the algorithm with deterministic sampling[3] -i.e. actual closest node- to obtain 128 samples for each product-time combination, and used the empirical distributions of the samples as predictions for the results in table 2. Model performance in all metrics is very poor compared to neural-net

[3] Future work will include ablations on the temperature parameter to see if it improves performance.

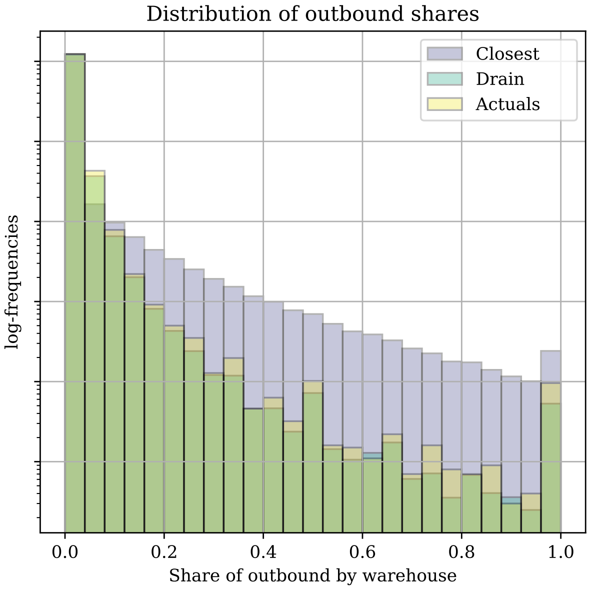

Figure 3: The log-frequencies of shares of outbound per warehouse. A share of 1 implies all outbound for a product-week is performed by a single warehouse. It is visible from the closest node distribution that it tends to concentrate outbound in few nodes, while the NN based Drain model tends to resemble the distribution of actual outbound.

based models. The reason is that the model outputs distributions of outbound units which are overly concentrated, i.e. all outbound is performed by a few warehouses, which are usually close to large cities where clusters of demand are present. Figure 3 is an exemplification of this fact.

Discussion and future work: We attribute the underperforming results of this model to a multitude of reasons. For starter, the data generation process concentrates all customer demand to have equal GPS coordinates in each region, thus resulting in one node always taking up all the demand for a particular region. More in general, closest node is a simplistic algorithm that does not take into account a multitude of factors that determines the F outbound decision, such as transportation costs, shipments costs, capacity, customer shipment consolidation and others. For these reasons we believe that closest node needs to improve and consider other variables before reaching the sophistication necessary to mimic F behavior. We're currently in the process of expanding the research on these baseline models, as there is clear value in good models whose properties and behavior are human interpretable.

F A model for Glance View conversion

We present here a model to sample the conversion of a glance view into an order. At a high level, the model assumes that glance views are independent and have conversion probabilities that are exogenous given a product, time and region combination. If we have estimates of the conversion probabilities, we can replay historic glance views with counterfactual promises to simulate orders.

We now define our choice of $\mathcal{O}$ and $\mathcal{S}$. Promises are grouped into three types: one-day (1d), two-day (2d), three or more days (3d+) and out-of-stock (-). Similarly, we group ship options into four types: one-day (1d), two-day (2d), three or more days (3d+) and no order (-). Thus we have $\mathcal{S} = \{1d, 2d, 3d+\}$ and $\mathcal{O} = \{1d, 2d, 3d+, -\}$.

We model each glance view as an independent sample from a multinomial distribution where promise $j \in \mathcal{S}$ has probability $p_{j,k}$ of converting to SO $k \in \mathcal{O}$. The model can be represented as a $|\mathcal{S}| \times |\mathcal{O}|$ matrix as in 12. Note that there are never ship-options available that are faster than the promise shown to the customer, and thus those conversion probabilities are always 0. Similarly, when the item is out-of-stock, the probability of no order is 1.

Table 12: A simple model for glance views conversions based on probabilities.

		Ship Option			
		No-Order	1d	2d	3d+
Speed	Out-of-Stock	1	0	0	0
	1d	$p_{1d,-}$	$p_{1d,1d}$	$p_{1d,2d}$	$p_{1d,3d+}$
	2d	$p_{2d,-}$	0	$p_{2d,2d}$	$p_{2d,3d+}$
	3d+	$p_{3d+,-}$	0	0	$p_{3d+,3d+}$

We estimate the elements of the matrix using historical data.