Simplifying Fulfillment Approximation with a Machine Learning Surrogate Model

by

Ahmed Abdelhak

and

Alex Wu

Submitted on May 5th, 2026 in Partial Fulfillment of the

Requirements for the Degree of Master of Applied Science in Supply Chain Management

at the Massachusetts Institute of Technology

As e-commerce growth drives customer expectations for faster and more flexible delivery, companies increasingly rely on distributed fulfillment networks to shorten the time between order placement and delivery. Evaluating the cost and flow implications of these networks typically requires high-fidelity simulation models that solve individual order-level fulfillment decisions across all feasible routes. The process is computationally intensive and limits rapid scenario-based analyses for strategic and tactical planning. This project develops a graph neural network (GNN) surrogate model that approximates the aggregate outputs of such a simulator, specifically total episode shipping cost and route-level fulfillment flows, without explicitly solving the optimization problem for every order. The final model achieves a test-set nonzero flow F1 of 0.860, and a weighted MAPE of 1.39% on episode total cost across 150 held-out episodes. The trained model produces a per-episode runtime that is approximately 40 times faster than the high-fidelity simulator. These results demonstrate that a simplified, machine learning-driven network flow model can serve as a practical tool for rapid what-if analysis across diverse demand and inventory scenarios. It can enable supply chain planners to evaluate network design alternatives at a fraction of the computational cost of traditional simulation approaches.

Capstone Advisor: Dr. Sarah Schaumann

Title: Postdoctoral Associate at the MIT Center for Transportation and Logistics

Table of Contents

1. Introduction

1.1 Background

In recent years, especially with the rapid growth of e-commerce, customer expectations for faster and more flexible delivery options have grown sharply. As Soetan (2025) points out, to stay competitive, many companies expand their fulfillment network, aiming to shorten the time between customer order submission and order fulfillment. While these strategies effectively increase the reachable service area and responsiveness, they also add substantial levels of operational complexity.

Practitioners and researchers explore ways to deal with increased complexity and make operations more efficient. As Aćimović and Graves (2014) describe, a typical fulfillment optimization problem involves deciding the source node (warehouse), intermediate nodes (fulfillment centers), and flow arcs (shipping routes and modes), so that an order with one or more stock keeping units (SKUs) can be delivered to its sink node (customer destination), with an objective of minimizing cost or obtaining maximum reward. Major retailers with a distributed network, such as Amazon, must decide quickly, for every order, which location should ship the requested SKUs and how. Since it cannot see future orders, each decision today affects future fulfillment options, making both daily decisions and the overall design of the network difficult to manage. A common approach to studying such dynamics is through computational modeling and simulation.

Building on this foundation, recent research conducted at the Intelligent Logistics Systems Lab at the Massachusetts Institute of Technology (MIT) developed a model that simulates online order fulfillment decisions in a distributed network. This model includes the implementation of a heuristic policy to guide individual SKU fulfillment decisions within an order. The heuristic computes all feasible fulfillment routes sequentially for every SKU and picks the lowest cost option. The resulting shipping costs and fulfillment decisions provide a reference point for assessing the cost implications of various scenarios.

1.2 Motivation

The simulation approach to solving fulfillment optimization problems relies on explicitly evaluating all feasible flows between network nodes to identify the cost-minimizing decisions. This leads to high computational complexity and makes scenario-based analyses time-consuming. In practice, however, operational (daily), tactical (weekly to monthly), and strategic (yearly) decisions are interconnected, and strategic choices do not require full visibility into every fulfillment action. Rather than simulating thousands of individual orders, planners often only need aggregate flow patterns and total cost estimates to support higher-level decisions.

The MIT research encountered a similar challenge when modeling a complex fulfillment network designed to produce insights for strategic planning. While the model accurately captured the cost and flow dynamics of the system, its computational intensity limited the ability to perform rapid evaluations with aggregate results across different scenarios or planning horizons.

This motivates the development of a surrogate model that can approximate aggregate fulfillment outcomes without solving the full optimization problem on an order level. Since the fulfillment system is naturally organized as a network of facilities, customer regions, and transportation lanes, graph neural networks (GNN) are a natural method to explore. Zhou et al. (2020) describe GNNs as neural models that capture graph dependence through message passing between connected nodes. Message passing describes how one node updates its information when receiving information from the nodes that are connected to it (Gilmer et al., 2017). This is relevant to our project because shipping cost and fulfillment

decisions are shaped by interactions across the fulfillment network, rather than by isolated variables alone. In addition, the simulator shows nonlinear behavior, since shipping cost includes both a fixed cost per shipment and a variable cost based on unit weight, while fulfillment decisions are sensitive to location-based features and carrier constraints. These characteristics make GNNs a promising direction for the surrogate modeling task.

1.3 Problem Statement and Key Questions

This project investigates a computationally lightweight approach to the complex fulfillment optimization problem faced by a company managing a network of warehouses that serve customers from different regions. The approach considers several key factors to formulate the approximation of the aggregate fulfillment results, including arc-based transportation costs (outbound and transshipment), node locations, transportation modes with various shipping capacities, and initial inventories.

The study seeks to determine whether a surrogate model can accurately approximate episode-based total shipping costs and aggregate flows produced by a high-fidelity simulator.

Accordingly, this research will be guided by the following key questions:

Research question1: Can a GNN-based surrogate network flow model accurately approximate the cost and aggregate order fulfillment decisions produced by a high-fidelity order fulfillment simulator?

Sub-research question: How does the surrogate model perform relative to the original optimization simulator in terms of cost accuracy, and computational speed?

Research question2: Which GNN approaches achieve high approximation accuracy and generalization?

1.4 Project Goals and Expected Outcomes

The project pursues the following objectives. Model development involves formulating a simplified network flow model that represents key structural elements of the complex fulfillment network, including supply and demand nodes, transportation routes, and capacity constraints. Surrogate model evaluation assesses performance along three dimensions: accuracy, the ability to reproduce total cost and aggregate flow patterns; generalization, consistency of performance under new or unseen demand scenarios; and computational efficiency, the reduction in solve time compared to the high-fidelity model.

By the end of this project, we expect to deliver a validated graph neural network modeling framework that accurately approximates fulfillment outcomes while significantly reducing computational requirements. To ensure robustness, we quantify the trade-offs between model simplicity, predictive accuracy, and processing speed.

2. State of the Practice

The central question of this project is how to approximate the aggregate outputs of a high-fidelity order fulfillment simulator, with a pre-set heuristic, using far less computational effort and processing time. To address this question, we reviewed three streams of research: existing approaches to order fulfillment optimization, classical network flow principles, and graph neural network supervised learning techniques for mapping complex system behavior.

2.1 Existing Solutions to Order Fulfillment Problems

Heuristic Approach

The order fulfillment optimization problem involves deciding which facility should ship each SKU and which transportation mode to use to fulfill each order at minimum cost while maintaining service requirements. Early work by Aćimović and Graves (2014) formalized an influential framework to solve order fulfillment optimization problems in a simplified way. They approximate the multi-SKU order fulfillment problem using a single-SKU linear program, combining the immediate shipping cost (a greedy reward) with a shadow price of the inventory state obtained by solving the linear program. This shadow price represents the potential value of saving one more unit of inventory in the warehouse for future demands. Their model improves cost performance and offers a scalable solution for large datasets because the dual values update infrequently. However, the key simplification is that each SKU is computed independently, and multi-SKU orders are handled by applying post-hoc adjustments to reduce split shipments. This method retains tractability but has limited effectiveness for multicommodity order fulfillment and stochastic ordering patterns. Consequently, it performs best for high-volume SKUs with deterministic demand patterns, and less effectively for stochastic, slow-moving demand.

Aćimović and Farias (2019) documented a similar approach of combining a primal solution and its dual price, but with a tweak to avoid demand forecasting. Their online policy updates dual prices after each order fulfillment. Their method is robust and effective toward SKUs with diverse ordering patterns. However, updating dual values after every order comes at a higher computation cost and reduces scalability, making this method less suitable for very large SKU catalogs. As Aćimović and Farias (2019) pointed out in their review, the key drawbacks of these approaches using dual prices lie in optimizing multi-SKU order fulfillment and transshipment (a shipment that goes through one or more intermediate destinations before reaching the final destination) decisions. These papers collectively reveal that multicommodity fulfillment optimization remains challenging. Because most approaches still decompose the problem into single-commodity models, important systemwide interactions may be lost. Forecast-free methods offer robustness but are often computationally heavy and difficult to scale. In addition, these online decision methods are very slow at providing an aggregated view of network flows, bottlenecks, or overall cost impacts, which are crucial for higher-level planning.

Network Capacity-Based Control Policies

Levin (2024) offers a complementary perspective by studying order fulfillment as a stochastic, multicommodity network control problem where congestion at nodes and arcs is a primary bottleneck. A maximum-pressure policy optimizes throughput by selecting arcs that relieve congestion-related pressure in the network. This approach gives a broader, system-level view than myopic cost-minimization heuristics. It also guarantees network stability and avoids service disruption, but does not directly minimize shipping cost. Levin's method is insightful for identifying bottlenecks and rebalancing inventories in the network, highlighting the importance of parameterizing flow capacities. But like other online guidance policies, it does not offer an aggregated view of shipping costs or network behavior, which is the central goal of our project.

Machine Learning Surrogates for Operational Models

A separate stream of recent work uses machine learning to approximate the outputs of computationally expensive operational models, an approach that overlaps closely with the goals of this project. Bengio et al. (2021) survey learning-based approximations of combinatorial optimization problems and identify three distinct paradigms: end-to-end learning that maps problem instances directly to solutions, learning to configure existing solvers, and learning surrogate objective functions that guide search. Our project falls into the first category. The motivation across all three paradigms is the same: when the same problem is solved many times under varying inputs, an amortized learned approximation often pays back its training cost within a small number of inferences.

Within supply chain operations specifically, prior work has applied learned surrogates to inventory allocation, last-mile delivery routing, and warehouse picking. The reported speedups vary widely depending on how expensive the underlying operational model is, but the pattern is consistent: surrogates that capture aggregate outputs (such as total cost or service level) typically achieve greater speedups than surrogates that try to reproduce every detail of the original solver's output. This pattern motivated our decision to focus the surrogate on episode-level aggregate metrics rather than reproducing the full per-order routing decisions exactly. The trade-off is documented quantitatively in Section 4.

2.2 Principles of Network Flow Problems

To understand how a surrogate model is a feasible approach to a complex problem, we reviewed fundamental network flow concepts. Assad (1978) provides a comprehensive view of multicommodity network flows (MCFPs). The paper points out that industrial distribution systems naturally map onto flows of multiple commodities (SKUs) across shared arcs with different shared capacity constraints, between source (warehouses), transport (fulfillment centers), and sink (customer) nodes. In its simplest form, a single-commodity flow problem with an objective function to minimize shipping cost can be written as:

$$\min \ \Sigma \, c_{ij} x_{ij}$$

$$\text{subject to}$$

$$x_{ij} \le u_{ij}, \ \forall (i, j) \in A, \ x_{ij} \ge 0$$

$$\Sigma \, x_{ij} \ge d_j, \ \forall j \in N_{demand}$$

where c_{ij} is the per-unit cost of shipping from node i to node j, x_{ij} is the flow on that arc, u_{ij} is the shipping capacity of this arc, and d_j is the demand at node j.

Assad (1978) emphasizes that real-world network design rarely requires exact optimal flows. Approximate flows are often sufficient instead of finding exact optimal solutions. This premise directly motivates our approach. We use modern supervised learning techniques to quickly approximate flow outputs then apply fulfillment costs to inform network design decisions.

2.3 Machine Learning Methods

Common Supervised Learning Methods

In order to make informed decisions on suitable machine learning methods, we reviewed the most common supervised learning principles. Drawing from the works of Bengio et al. (2021), Shalev-Shwartz and Ben-David (2014), and Goodfellow et al. (2016), Table 1 summarizes selected relevant classes of learning methods.

Model Type	Strengths	Weaknesses
Linear / Generalized Linear Models	Fast, interpretable; low risk of overfitting	Cannot capture nonlinearity; prone to underfitting by over-simplification
Tree-based Models (Random Forests, Gradient Boosting)	Handle tabular data well; capture nonlinear relationships; require little scaling	Can overfit with small datasets; can be computationally costly to tune
Neural Networks	Highly flexible; perform well with complex nonlinear functions; scalable	Require large datasets; sensitive to hyperparameters

Table 1. Comparison of Supervised Learning Methods Relevant to Surrogate Fulfillment Modeling.

Graph Neural Network Methods

GNNs appear to be a promising family of methods for this project because they intuitively model the fulfillment network structure. A GNN consists of nodes, like the facilities and customer regions, as well as edges, like the transportation lanes. Battaglia et al. (2018) explain that graph networks are designed for problems where entities and the relations between them are central to the task. In our project, the simulator determines the shipping costs and fulfillment decisions by computing the interactive effects of the node and edge features, such as relative distances and connecting carriers' capacities. The underlying relational mechanisms also show nonlinear behavior, because the shipping cost depends on both a fixed cost per shipment and a variable cost proportional to the shipment weight. These combined effects may be difficult for linear and tree-based models to capture consistently. For this reason, GNNs are a natural method to explore for building a surrogate model of the fulfillment system.

GNNs are also attractive because they can combine several types of information within a single model. In a graph, nodes represent the locations in the system, and edges represent the connections between them. Battaglia et al. (2018) explain that graph networks can not only use local inputs like the node and edge features, but they also can learn global attributes that describe the system as a whole. This flexibility may unveil valuable underlying information to learn, such as the demand distribution and shipment lane density. In practical terms, this allows the model to detect patterns from both the global structure of the fulfillment network and the individual order characteristics.

Among the candidate GNN models, a message passing neural network (MPNN) appears to be relevant. Gilmer et al. (2017) describe an MPNN as a graph learning model that allows its nodes to pick up information sent from neighboring nodes and update their own attributes. After the message passing round, the model converts the updated network information into a final prediction. For this reason, MPNNs are very strong in learning and predicting relational information, which closely resembles the fulfillment decisions.

A second option is an edge-aware GNN. This type of model relies more on using the information attached to each transportation lane when learning from the network, rather than relying mainly on updating the information of the connecting nodes (Simonovsky and Komodakis, 2017). This is relevant to the present project because transportation lanes may differ in cost, capacity, distance, and feasibility to ship specific SKUs. If these lane attributes have a strong influence on shipping cost and fulfillment decisions, a model that responds directly to edge information may be more suitable than one that relies mainly on the network structure itself.

3. Methodology

3.1 Data Generation

The simulator acts as the ground-truth generator for learning. Given a defined operational time window, the simulator produces an episode: a sequence of incoming orders, the heuristic policy's fulfillment decisions on each order, the resulting shipping costs, and the inventory state evolution that links one order to the next. Each episode is exported as a structured record containing the network topology, node and arc attributes for that episode, the orders, and the fulfillment plan applied to each order.

The training corpus consists of many episodes generated by repeated runs of the simulator under varied initial conditions. Variation in initial inventory levels, demand sampling, or carrier availability between episodes ensures that the corpus covers the operational variability the surrogate is expected to generalize over. Each order within each episode becomes one supervised learning sample, with inputs derived from the network state at the time the order arrives and targets derived from the simulator's fulfillment decision for that order.

3.2 Graph Construction

Each order is converted into a graph object suitable for a deep learning library such as PyTorch Geometric (PyG). The graph has three components.

Nodes represent fulfillment facilities and customer demand regions. Node features include any static attributes that influence routing decisions in the simulator, such as facility capacity, initial inventory levels, and geographic location. Each node feature vector is denoted $x_i \in \mathbb{R}^{d\_n}$. An order-specific destination indicator is appended at load time so the model can identify which region the order targets.

Edges are directed and represent feasible shipment arcs in the network, both transfer arcs between facilities and outbound arcs from facilities to customer regions. Edge features include transportation cost parameters (typically fixed cost per trip, variable cost per unit weight, and vehicle capacity), distance, and shipping time. Each edge feature is denoted $e_{ij} \in \mathbb{R}^{d\_e}$.

Order line features encode the SKU identities and the line-level masses that make up the order, preserving the multi-line composition.

Feasibility Mask and Per-Order Edge Features

A key preprocessing step computes, for each order, which arcs are structurally eligible to carry any flow for that order. An outbound arc is feasible only when its destination region matches the order's destination and its carrier is permitted to handle at least one SKU in the order. A transfer arc is feasible only when its destination facility has at least one outbound arc that is itself feasible for the order. The resulting feasibility mask is stored per order, alongside an edge coverage fraction (the fraction of the order's total weight that each arc could carry given any per-arc SKU restrictions) and a binary full-coverage flag. These per-order scalars are appended as additional columns to the static edge features at load time.

Without this preprocessing, the classification head would have to distinguish active arcs from inactive arcs over a candidate pool that contains every arc in the network. With the mask, the candidate pool per order is reduced to a small subset of the network. The exact reduction depends on the network and the carrier-SKU coverage rules of the dataset; the goal is to keep the true-positive capture rate near 1.0 (every arc that actually carries flow in the simulator's plan should remain in the candidate pool).

3.3 Surrogate Architecture

The surrogate model is implemented as a multi-stage GNN with three functional blocks: an order embedding layer, a graph attention backbone, and prediction heads for arc-level flow and order-level cost. The purpose of describing the architecture as a sequence of blocks is to make it portable across datasets: each block can be sized to match the complexity and feature dimensionality of a different deployment without changing the overall structure.

Order Embedding

An order embedding layer summarizes the multi-line composition of each order into a single fixed-size vector. SKU identities are mapped to learned vectors via an embedding table, weighted by the line mass, and aggregated (typically averaged) across lines to produce an order vector. This vector is broadcast to every node in the graph and concatenated with the node features to create the initial node state:

$$h_i^{(0)} = W_n \cdot [\, x_i \,;\, \text{order\_vec} \,]$$

Edge features are used directly without an additional projection, since they already include both the static lane attributes and the per-order feasibility scalars described in Section 3.2.

Message Passing Backbone

The backbone consists of stacked graph attention (GAT) convolution layers, which let each node's representation be updated as an attention-weighted aggregation of its neighbors' representations and the corresponding edge features:

$$h_i^{(\ell+1)} = \sigma(\ \phi(\ h_i^{(\ell)}, h_j^{(\ell)}, e_{ij}\ )\ )$$

where ϕ is implemented through learnable attention-weighted transformations, and σ is a nonlinear activation. Stacking multiple layers allows the model to capture multi-hop interactions corresponding to multi-stage fulfillment such as transshipment effects. Dropout and residual connections are applied at each layer for regularization.

According to Brody et al. (2022), attention is a mechanism that lets a neural network learn how much each input should influence a given output, by assigning a learned weight to each input rather than treating them equally. An attention head is a single calculation of this kind. Running several attention heads in parallel and combining their results helps the model pick up on different patterns at the same time. A graph attention network (GAT) applies this idea to graphs. Each node looks at its neighbors, decides which ones matter most for its task, and updates itself. The GATv2 variant used here makes the attention computation more expressive than the original GAT formulation by allowing the ranking of which neighbors matter most to depend on the querying node. GATv2 is an edge-aware variant of the message passing framework discussed in Section 2.3, combining the message-passing structure of an MPNN with the edge features.

Flow Prediction Heads

For each candidate arc, an arc representation is formed by concatenating the final node embeddings at the two endpoints with the arc features:

$$z_{ij} = \text{Concat}(\ h_i^{(L)}, h_j^{(L)}, e_{ij}\ )$$

Two separate prediction heads are applied, each implemented as a multi-layer perceptron (MLP), a small feed-forward neural network. The node embeddings used here are the contextualized representations after all message-passing layers have run, not the raw input features. Re-including the arc features at this stage gives the prediction head direct access to that specific arc's properties. A classification head produces the probability that the arc carries any flow:

$$\hat{p}_{ij} = \sigma(\ \text{MLP}_{cls}(\ z_{ij}\ )\ )$$

A magnitude head predicts the quantity shipped on the arc:

$$\hat{m}_{ij} = \text{softplus}(\ \text{MLP}_{mag}(\ z_{ij}\ )\ )$$

Within the forward pass, the feasibility flag is used to mask the classification logits on infeasible arcs so that their predicted activation probability is effectively zero. During decoding, an arc is considered active if its predicted probability exceeds a chosen decision threshold; the predicted flow on an active arc is the magnitude head's output.

Cost Prediction with Analytical Base

Because shipping cost is a deterministic function of flow given known cost parameters, the cost head is structured as a residual on top of an analytically computed base cost. For each arc with predicted flow $\hat{y}_{ij}$, the base cost contribution is the simulator's own cost formula, typically a fixed charge per vehicle trip plus a variable charge proportional to shipment weight:

$$\hat{c}_{analytic} = \Sigma\ [\ fixed_{ij} \cdot \text{trips}(\ \hat{y}_{ij}, cap_{ij}\ ) + var_{ij} \cdot \hat{y}_{ij}\ ]$$

Pooling happens at the order level here. For each order, the model averages the node embeddings from that order's graph into a single summary vector, which the residual head uses to predict a cost correction

for that order. Episode-level cost is then obtained by summing the predicted costs across all orders in the episode. A small residual head, fed by a pooled graph embedding, learns a correction on top of this base cost:

$$\hat{c} = \text{softplus}(\hat{c}_{analytic} + \text{MLP}_{resid}(\text{pool}(h)))$$

This design grounds the cost prediction in the simulator's own cost formula, so the network does not have to relearn deterministic arithmetic. The neural residual is reserved for effects that the analytical base cannot capture, such as discrete trip rounding errors propagated through the predicted flows.

3.4 Training Procedure

The training corpus is split at the episode level into training, validation, and test partitions. Episode-level splitting ensures that no single episode is shared across partitions, preventing data leakage at evaluation time.

The training objective combines an arc-level flow loss with an order-level cost loss. The arc-level flow loss has two components, both computed only over the feasible arcs of each order. The classification loss is binary cross-entropy (BCE) with positive-class upweighting computed from the feasibility-restricted class balance:

$$L_{cls} = \text{BCE}_{masked}(\hat{p}_{ij}, 1[y_{ij} > 0])$$

The magnitude loss is mean squared error (MSE) on log-transformed flows plus a Huber term on the raw predicted shipment weight for positive-flow arcs:

$$L_{mag} = \text{MSE}(\log(1 + \hat{m}), \log(1 + y)) + \lambda_{kg} \cdot \text{Huber}(\hat{m} - y)$$

The cost loss is mean squared error on the predicted total cost:

$$L_{cost} = \text{MSE}(\hat{c}, y_{cost})$$

The total objective combines these with tuning weights:

$$L = w_{flow} \cdot (L_{cls} + L_{mag}) + w_{cost} \cdot L_{cost} + \lambda_{resid} \cdot \|r\|^2$$

Standard practices apply: AdamW or similar adaptive optimizer, a warmup-then-cosine-decay learning rate schedule, gradient clipping, mixed-precision arithmetic where supported, and early stopping based on validation loss.

3.5 Treatment of Temporal Dynamics

The model's inputs fall into two groups. Static inputs do not change across orders within an episode and include the network topology, edge attributes, and start-of-episode inventory levels. Order-specific inputs change with every order and include the SKU lines and masses, the destination region, and the per-order feasibility values from Section 3.2.

In a real fulfillment system, inventory and capacity availability also change as orders are processed, so the optimal routing for a later order can differ from an earlier order with otherwise identical features. The framework treats each order as an independent sample conditioned on the start-of-episode network state, without passing updated inventory from one order to the next. This is a deliberate simplification that prioritizes scalability over capturing within-episode dynamics. Section 4.7 discusses the implications, and Section 5.2 describes possible extensions.

3.6 Inputs and Outputs at Inference Time

At inference time, the surrogate takes three groups of inputs: the network description (static node and edge features), the order specification (SKU lines with masses and the destination region), and the computed feasibility information described in Section 3.2.

The model produces three outputs per order: a probability score per arc, a magnitude estimate per arc (predicted shipment weight), and a scalar predicted total shipping cost. From these, downstream applications can construct the metrics they need: arc activation by thresholding the probabilities, episode-level cost by summing per-order costs, and aggregated lane utilization by summing per-order flows.

3.7 Evaluation Metrics

Model performance is evaluated on the held-out test partition along three dimensions. Cost accuracy is reported as the coefficient of determination (R^2), mean absolute error (MAE), root mean squared error (RMSE), and weighted mean absolute percentage error (wMAPE) at both the order level and the episode level (aggregated from the orders within each test episode). Flow prediction quality is reported as precision, recall, and F1 score for nonzero-arc identification, together with R^2 on predicted versus true flow magnitudes for active arcs. Calibration is assessed through decile plots of predicted versus true cost. Computational efficiency is evaluated as per-episode wall-clock inference time compared against the high-fidelity simulator on equivalent hardware.

4. Results and Discussion

4.1 Implementation and Design Choices

This section documents the specific design choices and dataset characteristics used to instantiate the framework of Section 3 for our experiments. The training corpus consists of 1,000 simulated episodes, each containing approximately 200 orders drawn from a catalog of 3,000 SKUs. The directed graph contains 1,016 arcs that span both transfer lanes between facilities and outbound shipping lanes from facilities to customer regions. Across all orders, the per-order feasibility mask reduces the candidate arc pool from 1,016 to roughly 26 on average, with a true-positive capture rate of 1.00 measured on the training data, meaning every arc that actually carried flow in the simulator's plan was retained in the candidate pool.

Embedding and Architecture Sizing

The SKU embedding table maps each of the 3,000 SKUs to a 32-dimensional learned vector. This dimension was chosen to give each SKU enough capacity to encode product-level patterns relevant to routing, such as size class and typical destination, without dominating the model parameter count. The destination region indicator is encoded as a one-hot vector over the 18 regions and concatenated with each node's feature vector at load time.

The graph attention backbone uses four GATv2 convolution layers with four attention heads each and a hidden dimension of 192. Four layers were selected to provide enough message-passing depth to capture the longest typical path in this network (a transshipment route has at most three hops: source facility, intermediate facility, destination region). Hidden dimension 192 was chosen by trial and error as the smallest value that did not bottleneck the multi-head attention. Dropout of 0.15 is applied at each layer.

After the backbone, two prediction MLPs (each two layers, hidden dimension 128) produce the classification logit and magnitude estimate per arc. The cost residual head is a single-layer MLP from a global mean-pooled graph embedding to a scalar correction.

Training and Decoding Settings

Training was carried out fully in PyTorch. The 1,000 episodes were split into 70% training (700 episodes, 139,991 orders), 15% validation (150 episodes, 29,997 orders), and 15% test (150 episodes, 29,994 orders). Training ran for 150 epochs with patience of 15 on the validation total loss, AdamW optimizer, learning rate 1e-3 with eight epochs of linear warmup followed by cosine decay, and gradient clipping at norm 1.0. The flow loss weights were w_flow = 2.0 with magnitude weight ratio 1.0 and an additional Huber penalty weight λ_kg = 0.05 on positive-arc kilograms. The cost loss weight was w_cost = 2.0, and the residual L2 weight was 1e-4. The classification decision threshold for decoding active arcs was selected from a grid search on the validation set; threshold 0.8 was chosen and is discussed in Section 4.3.

Hyperparameter Selection Rationale

Key hyperparameter choices were guided by small-scale ablations on a held-out subset of the training data and by training-loss dynamics. The number of GAT layers (four) matches the longest fulfillment path in this network (a three-hop transshipment route), ensuring that information can travel from any source to any destination within the receptive field of the final node embeddings. Three layers underfit validation flow F1 by approximately three percentage points; five layers offered no measurable gain. The hidden dimension (192) was selected by sweeping {96, 128, 192, 256}; validation loss decreased materially up to 192 and was effectively flat beyond, so 192 keeps the parameter count moderate (around 547,000) while preserving the validation ceiling.

The flow positive-class weight is computed automatically from the feasibility-restricted class balance (about one in 25 feasible arcs carries flow), avoiding manual tuning and adapting if the dataset changes. The flow and cost loss weights were set equal (w_flow = w_cost = 2.0) after early runs showed that unbalanced weighting caused one head to underfit. The arc-activation decision threshold was selected post-training by a fine-grained sweep on the validation set (Section 4.3), which preserves the option to recalibrate the cost-precision trade-off in deployment without retraining.

4.2 Training Behavior

The final model completed 150 training epochs without triggering the early-stopping patience window. Figure 1 shows the total training and validation losses. Both curves declined steadily through training, with train loss dropping from 33.3 at epoch 1 to 9.3 at epoch 150, and validation loss dropping from 21.2 to 5.96 over the same window. Validation loss continued to decrease through the end of training, indicating that further training under the same schedule could yield incremental improvements.

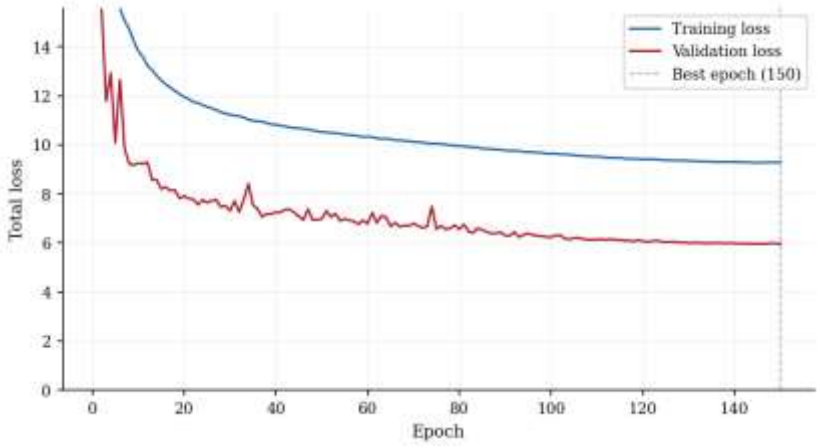

Figure 1. Total Training and Validation Loss Across 150 Epochs.

Figure 2 separates the three major loss components. The flow classification loss converges rapidly within the first ten epochs, consistent with the feasibility mask making the activation problem relatively easy to learn. The flow magnitude loss declines more gradually throughout training. The cost prediction panel shows a notable behavior worth explaining: the train and validation curves sit at clearly different levels (train near 1.4, validation near 0.05) rather than converging to the same value as in the other two panels.

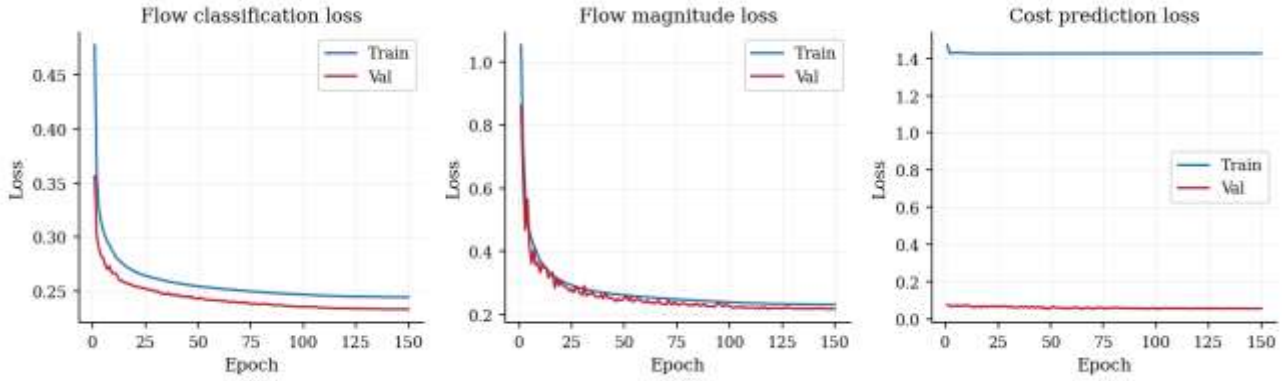

Figure 2. Loss Component Breakdown ver 150 Epochs. Flow Classification (left) and Flow Magnitude (middle) curves converge to similar values for train and validation. The Cost Prediction Curve (right) shows train and validation on different levels because of the trip-counting approximation explained below.

The cost head predicts cost by applying the simulator's cost formula to the predicted flow on each arc. The formula uses a discrete operation. The number of vehicle trips on an arc is the ceiling of the predicted flow divided by the arc capacity, rounded up to the next whole number. This rounding step has zero gradient almost everywhere, which would block the gradient flow needed for training.

To work around this, the model uses a smoothed approximation during training. It replaces the ceiling with the raw real-valued ratio, which keeps gradients flowing. At evaluation time, the model switches back to the exact ceiling so its outputs match the simulator. As a result, the cost loss reported during training is measured against a smoothed proxy, while the cost loss reported during validation uses the exact formula. The two values are on different scales and cannot be compared directly. The validation curve is the meaningful indicator of cost prediction quality, since it uses the same formula as the simulator and as the final test-time evaluation. It declines steadily across training and ends below 0.06.

4.3 Flow Prediction

On the held-out test set of 150 episodes (29,994 orders), the model achieves an active-arc F1 score of 0.860 at the selected threshold of 0.8. Precision is 0.860 and recall is 0.859, indicating balanced performance across both failure modes. The predicted fraction of active arcs (0.603%) closely matches the true fraction (0.604%), meaning the model does not systematically over- or under-activate arcs at this operating point.

Figure 3 shows how performance varies across thresholds. The F1 score peaks around threshold 0.7 to 0.8, and threshold 0.8 minimizes order-cost MAE while keeping F1 close to its peak. Thresholds below 0.7 sharply inflate the number of predicted active arcs, producing false positives that degrade cost accuracy. Thresholds above 0.9 drop recall and again increase cost error.

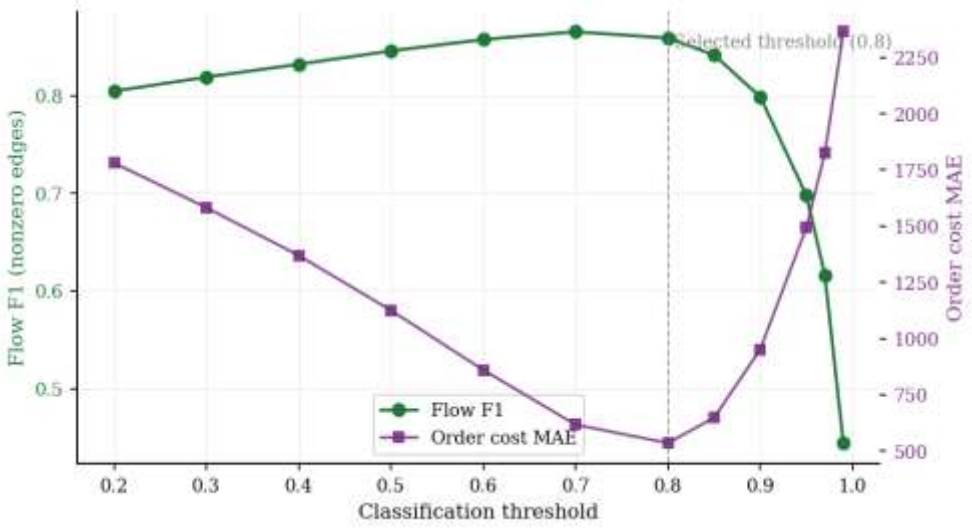

Figure 3. Flow F1 (left axis, green) and Order Cost MAE (right axis, purple) across Classification Thresholds on the Validation Set. The selected operating threshold is 0.8.

Threshold Stress Test

To assess the sensitivity of cost predictions to the chosen threshold, we evaluated the same trained model with the threshold lowered to 0.5, a common default for binary classifiers. Table 2 compares the two operating points on the test set.

Metric	Threshold 0.8 (selected)	Threshold 0.5 (default)
Flow F1 (nonzero arcs)	0.859	0.846
Predicted active-arc rate	0.603%	0.786%
Order cost MAE	536	1,125
Episode cost MAE	8,696	221,498
Episode cost wMAPE	1.29%	32.79%

Table 2. Threshold stress Test: cost accuracy at the selected threshold (0.8) versus a default value (0.5).

The F1 difference between 0.5 and 0.8 is small (0.846 vs 0.859), but the cost gap is dramatic. Episode-level wMAPE rises from 1.29% to 32.79%. The reason is that at threshold 0.5, the model activates roughly 30% more arcs per order, and each spurious activation contributes a fixed cost per vehicle trip in the analytical cost formula. Spurious activations therefore inflate the predicted cost in a way that compounds across orders within an episode. The selected threshold of 0.8 effectively trades a small amount of recall to suppress these false-positive trip charges, which is the right trade-off for the cost-prediction objective.

4.4 Cost Prediction

Order-level cost prediction on the test set achieves R^2 of 0.655, with a MAE of 537 cost units per order against a mean true cost of 3,376. Weighted MAPE is 15.9%. Episode-level cost prediction, aggregated across the approximately 200 orders within each test episode, achieves R^2 of 0.653, MAE of 9,387 cost units, and weighted MAPE of 1.39%. Figure 4 shows both sets of predictions against the true values.

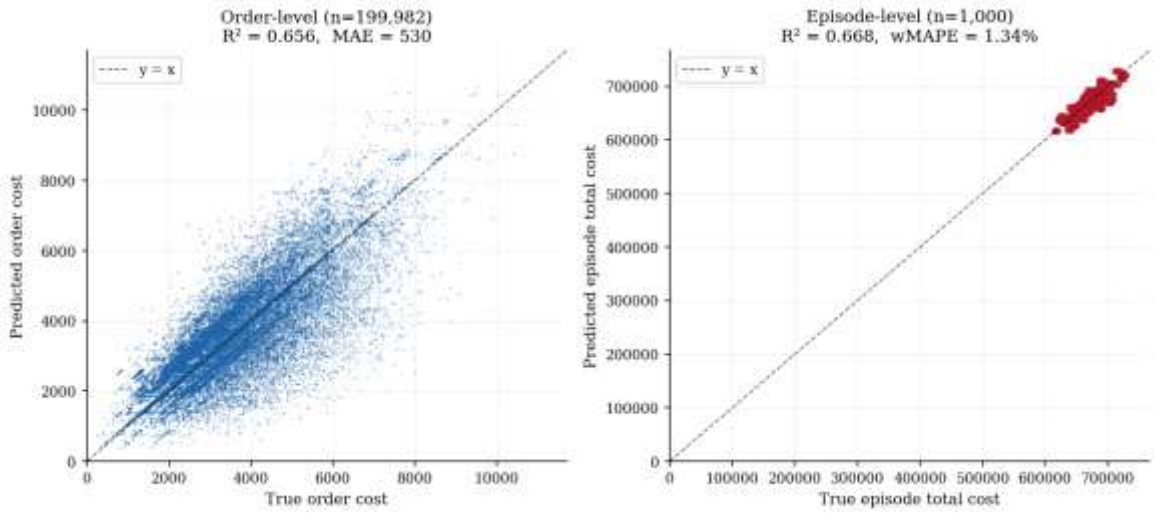

Figure 4. Predicted Versus True Cost on the Test Set. Left: Order Level across 29,994 Orders. Right: Episode Level across 150 Episodes.

The two scatter plots tell complementary stories. At the order level, there is noticeable spread around the y = x line, especially in the mid-range where most orders cluster. At the episode level, most of that order-level noise averages out and the predictions track the true totals closely. The 1.39% weighted MAPE at the episode level is particularly relevant for strategic planning applications, where the outputs of interest are aggregated cost impacts rather than individual order outcomes.

Figure 5 examines predicted versus true lane flows aggregated across all orders in each episode, restricted to the 856 of 1,016 arcs that carried any flow across the test set. The episode-level lane flow R^2 is 0.944, indicating that the model correctly identifies both which lanes activate and how much total volume passes through each active lane over an episode.

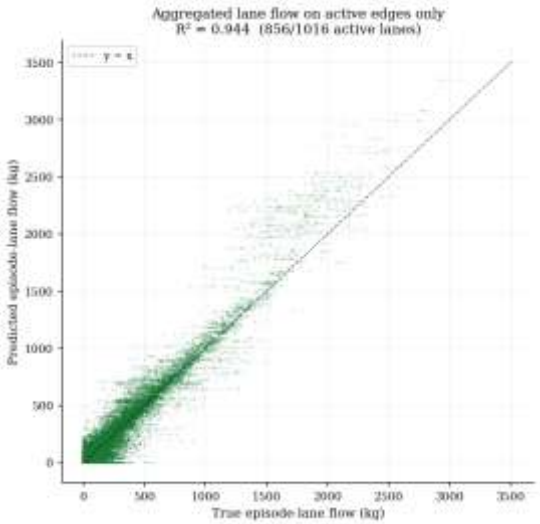

Figure 5. Predicted Versus True Aggregated Lane Flow, Summed across Orders within Each Episode. Only active lanes (with any true flow in the test set) are shown.

Figure 6 examines calibration across cost magnitudes. The model is well-calibrated in the middle deciles. In the lowest decile it under-predicts by about 384 units (1,615 vs. 1,999, a 19% under-estimate), and in the highest decile it over-predicts by 604 units (6,441 vs. 5,837). This pattern (under-prediction at low

extremes, over-prediction at high extremes) is common in regression models trained with squared-error objectives.

Two effects combine to produce this pattern. The first is regression to the mean: squared-error losses penalize large residuals quadratically, pulling predictions toward the global average. The second is dataset structure: cost extremes correspond to rare operational scenarios (very low-cost orders are small single-lane shipments; very high-cost orders fragment across multiple transshipment paths), so both tails have fewer training examples than the middle deciles. We did not pursue corrections such as a Huber-style cost loss or oversampling the extremes because the episode-level metric, which is what strategic planning consumes, is dominated by the middle deciles where calibration is already strong.

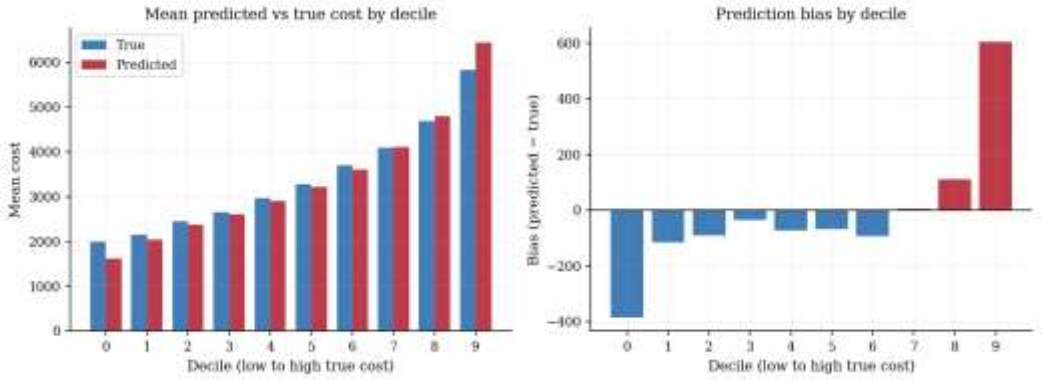

Figure 6. Order Cost Calibration by Decile. Left: Mean Predicted vs. Mean True Cost per Decile. Right: Signed Bias per Decile.

4.5 Comparison with the Heuristic Baseline Model

To establish a reference point for the surrogate, we constructed a simple heuristic baseline model. The baseline differs from the simulator's ground-truth heuristic in one important way: it operates under the same constraints as the surrogate. Specifically, the baseline does not have access to the within-episode inventory state, so it cannot adapt later orders to depleted inventory the way the simulator's ground-truth heuristic does. The simulator's ground-truth heuristic, by contrast, processes orders sequentially and rebuilds the feasible-route set after each order based on the updated inventory. Comparing against this baseline is more meaningful than comparing against the simulator itself, because it isolates the contribution of the GNN beyond what any rule-based method could achieve under the same input restrictions.

For each order, the baseline routes every SKU from the cheapest available source node and carrier, evaluating both direct and transshipment options under the start-of-episode inventory state. Table 3 summarizes the surrogate's test-set performance side by side with the heuristic.

Metric	GNN Surrogate	Heuristic Baseline
Order cost R^2	0.655	0.60
Order cost MAE	537	745
Order cost wMAPE	15.9%	21.9%
Flow F1 (nonzero arcs)	0.860	~0.39
Episode cost R^2	0.653	−1.31
Episode cost MAE	9,387	34,563
Episode cost wMAPE	1.39%	5.13%

Table 3. Test-set Performance of the GNN Surrogate Versus the Heuristic Baseline.

The surrogate outperforms the heuristic across every metric reported. On order-level cost, the gap is meaningful but moderate (R^2 0.655 vs 0.60), because the heuristic's deterministic cheapest-cost routing provides a reasonable baseline for individual-order cost estimation. On route identification, the surrogate's F1 of 0.860 substantially exceeds the heuristic's approximately 0.39. The heuristic is constrained to route every SKU to a single cheapest option and therefore fails to reproduce the mix of routes the simulator actually uses. The surrogate, by contrast, learned route-selection patterns from training data, including the sub-optimal splits that the simulator produces when inventory is depleted.

The most striking difference appears at the episode level. The heuristic's episode cost R^2 of −1.31 looks counterintuitive given that its order-level R^2 (0.60) is only modestly below the surrogate's value. Two effects led to this outcome at the episode level even though order-level R^2 is not bad.

The heuristic's predictions are systematically biased rather than randomly noisy. By always selecting the cheapest nominal route under the start-of-episode inventory state, the heuristic underestimates cost because the simulator's ground-truth heuristic, which has access to the evolving inventory, is forced to use more expensive substitutes once the cheapest routes are exhausted. The heuristic's order-level errors therefore have a consistent negative sign across orders. Biased errors do not cancel under aggregation the way unbiased errors do: random zero-mean errors summed across 200 orders would grow by only about $\sqrt{200} \approx 14$, but consistently signed errors accumulate linearly and grow by a full factor of 200. The cumulative under-prediction is large enough to exceed the across-episode variance of total cost, which is what produces the negative R^2.

The surrogate avoids this failure mode because it learned route-selection patterns from the simulator's actual fulfillment plans, which already reflect the substitutions caused by inventory depletion. Its order-level errors are noisier than the heuristic's but unbiased on average, so they cancel under aggregation. This is why the surrogate's episode-level metrics (R^2 0.653, wMAPE 1.39%) closely track its order-level metrics (R^2 0.655, wMAPE 15.9%) while the heuristic's metrics diverge sharply between the two scales.

4.6 Computational Efficiency

Running-time performance was benchmarked against the simulator on identical hardware using a 200-order episode as the unit of comparison. The simulator, running its full end-to-end pipeline (order generation, heuristic decision-making, inventory updates, and cost accounting), completes one 200-order episode in a mean of 1,972 ms over four timed episodes, with a standard deviation of 58 ms. Per-order simulator time is 9.86 ms.

The surrogate's inference cost is mainly driven by graph attention operations. Table 4 summarizes the comparison.

Component	Time per 200-order episode
Simulator	1,972 ms
GNN surrogate	~50 ms
GNN surrogate vs. simulator	~40× reduction

Table 4. Wall-clock Run Time Comparison.

The projected 40× speedup is meaningful for the intended use case. At the simulator's speed, evaluating 10,000 what-if scenarios at 200 orders each takes roughly 5.5 hours. At the projected surrogate speed, the same evaluation takes approximately 8 minutes. This order-of-magnitude improvement enables interactive scenario screening that is not practical with the simulator alone.

4.7 Limitations

The first limitation is that the model does not observe real-time inventory evolution. Each order is processed as if it arrives at the start of a fully stocked day. In practice, facility stock depletes as the episode progresses, and later orders are routed differently as a result. The feasibility mask captures static lane feasibility but not the dynamic availability of inventory at each source. This is the most likely explanation for the order-level noise in Figure 4 and the tail behavior in Figure 6.

The second limitation is that the model aggregates all SKU lines in an order into a single embedding before predicting arc-level flows, so information about which SKU lines contribute to which route is lost. The magnitude head consequently learns a statistical average across orders with similar aggregate profiles, rather than a deterministic quantity. We retained this design to keep the prediction unit at the order level, which simplified training. Moving to a per-SKU-line prediction unit is the most promising direction to push both magnitude and cost accuracy further.

The third limitation is that the reported accuracy reflects structural properties of this synthetic dataset that may not hold on other networks. Flow volume is highly concentrated (the top 6% of arcs carry 50% of total flow), and the feasibility mask narrows the candidate pool from 1,016 to roughly 26 per order. These properties make the prediction task easier: the model rarely has to distinguish between arcs with similar utilization or choose among many plausible options. A real-world network with broader carrier-SKU coverage or more uniform flow distribution would present a noisier signal.

5. Recommendations

The results in Section 4 support both near-term uses of the current surrogate and clear directions for continued development.

5.1 Near-Term Use

The current model may be suitable for two applications. The first is rapid scenario screening. With episode-level cost wMAPE of 1.39% and a projected 40× speedup over the simulator, the surrogate can evaluate large numbers of demand, network, or capacity scenarios at a cost and time budget that allow interactive exploration. The simulator can then be reserved for validating the handful of scenarios that the surrogate identifies as most promising or most uncertain.

The second is transportation lane analysis. An F1 of 0.860 on nonzero-arc identification, combined with a lane-flow R^2 of 0.944 at the episode level, indicates which lanes activate under a given scenario and roughly how much volume each lane carries, supporting carrier capacity planning and bottleneck identification. The simulator remains preferable for analyses that require detailed per-order outcomes, such as high-value-order analyses, SLA assessments, or inventory-replenishment studies; the surrogate's order-level R^2 of 0.655 is usable for ranking and screening but not for precise per-order conclusions.

5.2 Future Work

The most promising next step is to incorporate inventory levels as input features on facility nodes, allowing the model to observe stock depletion within an episode and address the temporal-dynamics limitation. A simpler near-term variant is to append a per-order index within the episode as a proxy for how much of the episode has elapsed.

Since the surrogate is a GNN, it operates on a variable number of nodes and edges in principle, but the current training is tied to one network. Future work should evaluate how well the model transfers when the underlying network changes (e.g., adding a facility or closing a lane). Training across a range of network variants during data generation would encourage topology-agnostic routing patterns and unlock network design use cases that the current model cannot support, such as evaluating new facility locations or carrier contract changes.

Another direction is multi-policy conditioning. The current surrogate is trained on data from a single fulfillment policy. Training a model that conditions on a policy identifier as an additional input would let it predict outcomes under multiple candidate strategies without separate retraining, which is valuable when evaluating proposed rule changes (such as adjusting transshipment thresholds or carrier preferences) before committing engineering effort.

Finally, operational deployment would benefit from a planning interface that lets users define scenarios (network changes, demand shifts, capacity adjustments) and view predicted episode cost, lane utilization, and bottleneck indicators in a single view, with the simulator triggered on demand for high-uncertainty scenarios.

References

Aćimović, J., & Graves, S. C. (2014). Making better fulfillment decisions on the fly in an online retail environment. *Manufacturing & Service Operations Management, 17*(1), 34–51.

Aćimović, J., & Farias, V. F. (2019). The fulfillment optimization problem. In *INFORMS Tutorials in Operations Research* (pp. 218–237). INFORMS.

Assad, A. (1978). Multicommodity network flows: A survey. *Networks, 8*, 37–91.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., ... Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks [Preprint]. arXiv. https://doi.org/10.48550/arXiv.1806.01261

Bengio, Y., Lodi, A., & Prouvost, A. (2021). Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research, 290*, 405–421.

Brody, S., Alon, U., & Yahav, E. (2022). How attentive are graph attention networks? In *Proceedings of the 10th International Conference on Learning Representations* (ICLR). https://openreview.net/forum?id=F72ximsx7C1

Dwivedi, V. P., Joshi, C. K., Luu, A. T., Laurent, T., Bengio, Y., & Bresson, X. (2022). Benchmarking graph neural networks. *Journal of Machine Learning Research, 23*(43), 1–48.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th International Conference on Machine Learning* (Vol. 70, pp. 1263–1272). PMLR. https://proceedings.mlr.press/v70/gilmer17a.html

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Levin, M. (2024). A real-time control policy to achieve maximum throughput of an online order fulfillment network. *Transportation Science, 58*(2), 434–453.

Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.

Simonovsky, M., & Komodakis, N. (2017). Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (CVPR) (pp. 3693–3702). IEEE.

Soetan, T. (2025). The role of fulfillment centers in e-commerce expansion in secondary markets. ResearchGate. https://www.researchgate.net/publication/390598411

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI Open, 1*, 57–81.