

Ask, Clarify, Optimize: Human–LLM Agent Collaboration for Smarter Inventory Control

Yaqi Duan

Leonard N. Stern School of Business, New York University
yaqi.duan@stern.nyu.edu

Yichun Hu

Johnson Graduate School of Management, Cornell University
yh767@cornell.edu

Jiashuo Jiang

Hong Kong University of Science and Technology
jsjiang@ust.hk

Inventory management remains a challenge for many small and medium-sized businesses that lack the expertise to deploy advanced optimization methods. This paper investigates whether Large Language Models (LLMs) can help bridge this gap. We show that employing LLMs as direct, end-to-end solvers incurs a significant “hallucination tax”: a performance gap arising from the model’s inability to perform grounded stochastic reasoning. To address this, we propose a hybrid agentic framework that strictly decouples semantic reasoning from mathematical calculation. In this architecture, the LLM functions as an intelligent interface, eliciting parameters from natural language and interpreting results while automatically calling rigorous algorithms to build the optimization engine.

To evaluate this interactive system against the ambiguity and inconsistency of real-world managerial dialogue, we introduce the Human Imitator, a fine-tuned “digital twin” of a boundedly rational manager that enables scalable, reproducible stress-testing. Our empirical analysis reveals that the hybrid agentic framework reduces total inventory costs by 32.1% relative to an interactive baseline using *GPT-4o* as an end-to-end solver. Moreover, we find that providing perfect ground-truth information alone is insufficient to improve *GPT-4o*’s performance, confirming that the bottleneck is fundamentally computational rather than informational. Our results position LLMs not as replacements for operations research, but as natural-language interfaces that make rigorous, solver-based policies accessible to non-experts.

Key words: Large Language Models, Inventory Management, Agentic Workflow, ERP Design, Decision Support Systems

1. Introduction

A recurring tension in management science lies between *normative optimality*—the decisions prescribed by formal models—and *descriptive reality*—the decisions practitioners actually make under

time pressure, incomplete data, and limited analytical support. Inventory management makes this tension vivid. Operations research has produced a mature toolkit for stochastic inventory control, including classical base-stock and (s, S) policies with sharp optimality guarantees under stylized assumptions (e.g. [Axsäter \(2006\)](#)), as well as modern approximate dynamic programming and deep reinforcement learning (DRL) methods for richer environments (e.g. [Madeka et al. \(2022\)](#)). In principle, these methods can reduce total cost (holding, ordering, and shortage) while improving service levels. In practice, however, many small and medium-sized retailers do not deploy them. Without dedicated analysts or data infrastructure, managers often rely on informal rules of thumb—“order when it looks low,” “double before holidays,” “round to a case-pack”—that are easy to apply but rarely calibrated to demand uncertainty, lead-time variability, or cost trade-offs.

This gap persists not primarily because the *optimal policy* is too hard to compute, but because standard optimization tools remain inaccessible to non-experts. A critical challenge lies in the intricacy of *problem formulation*. Even a basic inventory model requires translating messy operational context into precise inputs: what constitutes a stockout (lost sales vs. backorders); how lead times behave (deterministic vs. stochastic, including supply disruptions); what the review cadence is; which constraints bind (cash, storage, minimum order quantities, case-pack rounding, delivery schedules); and what objective proxies are acceptable (service-level targets, penalty costs, fill-rate). Much of this information is not stored in clean tables—it lives in human memory and informal language. This highlights an interpretive barrier: converting qualitative business narratives into a structured, model-consistent specification, and then converting the model output back into an operational routine that a manager can execute and trust.

Recent advances in large language models (LLMs) create a compelling opportunity to reduce this bottleneck, but also introduce a subtle trap. A natural first idea is to *directly call the LLM* as an end-to-end decision-maker: describe the business in natural language and ask, “What should I order?” This often produces fluent, confident recommendations. Yet fluency is not the same as correctness. Inventory control is a stochastic control problem where small structural mistakes—misinterpreting lead time, conflating lost sales with backorders, mishandling uncertainty, ignoring capacity or service constraints—compound into recurring costs. Moreover, LLMs are not designed to guarantee feasibility, respect Bellman optimality, or provide calibrated probabilistic reasoning. As a result, a direct LLM call can yield policies that are *plausible but systematically suboptimal*, or even internally inconsistent. In other words, naïvely using an LLM as the solver risks paying a “hallucination tax”: the efficiency loss induced when decision quality is limited by unconstrained language-model reasoning rather than grounded optimization.

The key insight of this paper is therefore not that LLMs can replace operations research, but that they can be used *more intelligently*: as an interface and orchestrator that makes rigorous optimization usable. We propose a hybrid, agentic decision-support framework that explicitly separates semantic interaction from mathematical optimization. Our system decomposes the pipeline into three roles: (i) a *Information Extraction Agent* that engages the user, surfaces missing information, and resolves ambiguity through targeted follow-up questions; (ii) an *Optimization Agent* that receives a structured specification and computes a policy using grounded operations research algorithms (e.g., (s, S) -style methods and DRL where appropriate); and (iii) an *Policy Interpretation Agent* that translates the computed policy back into operational guidance—what to order and when—while explaining assumptions, checking feasibility against stated constraints, and presenting actionable summaries in the user’s own language.

This separation is not merely an engineering choice; it is an epistemological one. It treats language models as tools for elicitation, structuring, and translation, while reserving policy computation for methods that can be verified, stress-tested, and improved with established theory. This design also clarifies what the “last-mile” problem in decision support really is. When strong algorithms already exist, the binding constraint is often the fidelity of the interface: how well it can elicit, stabilize, and formalize the problem instance that the solver requires. In our framework, the LLM acts as a rationality prosthetic—a front end that sanitizes inputs, makes assumptions explicit, and aligns business narratives with model structure—while the optimization back end remains the engine of efficiency. Rather than pursuing a one-size-fits-all, end-to-end Generative AI solver, we argue for a dual-engine approach in which LLMs unlock the rigor of management science for users who cannot otherwise access it.

To rigorously evaluate this dual-engine architecture, however, we face a methodological challenge. Interactive decision-support systems are difficult to evaluate at scale because real user inputs are noisy, inconsistent, and expensive to collect repeatedly. Static benchmarks miss the central difficulty: the agent must ask for missing parameters, handle contradictions, and converge to a well-posed model through dialogue. To enable controlled, reproducible evaluation, we introduce a *Human Imitator*: a language model fine-tuned on more than hundreds of real human-machine dialogues. The Human Imitator serves as a scalable “digital twin” of a boundedly rational small-business owner, reproducing the ambiguity, incompleteness, and occasional inconsistency characteristic of real managerial inputs. This allows systematic stress-testing of interactive systems without the logistical and financial burden of large human-subject trials.

1.1. Our Main Contributions

Our study makes four main contributions to the design and evaluation of LLM-based decision support for stochastic inventory control.

Performance quantification: Our hybrid agentic system reduces policy costs by 32.1% relative to a baseline in which *GPT-4o* is used as an interactive end-to-end solver. This gap provides a concrete estimate of the *hallucination tax*: the efficiency loss incurred when policy computation relies on unconstrained language-model reasoning rather than grounded stochastic optimization.

Drivers of Performance: We disaggregate performance to identify the specific regimes where agentic support yields the highest marginal value. Our analysis reveals that while the framework is distribution-agnostic (robust to demand shape), its advantage scales with *complexity* and *economic stakes*. The performance gap widens significantly under longer lead times and in high-penalty, high-flexibility regimes. This observed “complexity premium” confirms that solver-backed architectures deliver outsized returns precisely where the financial consequences of imprecise heuristics are most acute.

Limits of prompt-based reasoning: We isolate the source of decision error by comparing the interactive end-to-end *GPT-4o* baseline against a “perfect information” counterfactual. Strikingly, providing ground-truth parameters to *GPT-4o* yields no statistical improvement. This identifies a hard “cognitive ceiling”: the performance bottleneck of LLMs is not *informational* (data extraction) but fundamentally *computational*. This confirms that prompt engineering cannot bridge the gap to stochastic optimality; rather, LLMs function better when architected to *orchestrate* rigorous solvers.

Behavioral simulation for interactive benchmarks: Methodologically, we address the scarcity of interactive benchmarks by establishing a Human Imitator as a scalable proxy for boundedly rational managers. By reproducing the inconsistency and ambiguity of real-world inputs, this approach allows us to move beyond static datasets and stress-test decision support systems against the realistic friction of human-machine interaction. Beyond inventory control, our scalable evaluation pipeline offers a general template for evaluating interactive LLM-based decision-support tools in other operational domains.

Collectively, these findings suggest that the true potential of Generative AI in operations lies in its ability to democratize expertise. By serving as an intelligent orchestration layer on top of existing analytical tools, LLMs can finally unlock the power of management science for the long tail of small business owners who have historically been left behind.

1.2. Related Work

Our work sits at the intersection of three distinct streams of literature: stochastic inventory control (specifically Deep Reinforcement Learning approaches), the application of Large Language Models (LLMs) to optimization, and the evaluation of interactive dialogue systems through user simulation.

1.2.1. Stochastic Inventory Control and Deep Reinforcement Learning. The theoretical foundations of inventory management are mature, anchored by the optimality of (s, S) policies for single-echelon systems with linear costs (Scarf 1960) and their efficient computation (Axsäter 2006). Interested readers may refer to textbooks such as Simchi-Levi et al. (2005) for further details. However, real-world complexities—such as lead-time variability, lost sales, and multi-echelon networks—often render exact dynamic programming intractable due to the curse of dimensionality. To address these high-dimensional state spaces, recent scholarship has pivoted toward Deep Reinforcement Learning (DRL). DRL, when combined with deep neural networks, effectively addresses the high-dimensional state and action spaces inherent in inventory control, alleviating the curse of dimensionality. Early studies demonstrated the feasibility of RL in inventory control. For instance, Oroojlooyjadid et al. (2022) utilized the Deep Q-network (DQN) to solve the beer distribution game, a widely studied supply chain simulation. Similarly, Gijsbrechts et al. (2022) employed the A3C algorithm to achieve heuristic-level performance, while Stranieri and Stella (2023) benchmarked multiple DRL methods, including A3C, PPO, and vanilla policy gradient (VPG), for inventory problems. Recent advancements have extended DRL to a variety of inventory control scenarios, such as managing non-stationary uncertain demand (Dehaybe et al. 2024, Park et al. 2023a), optimizing multi-product systems (Sultana et al. 2020, Selukar et al. 2022), and handling diverse product types (Meisher et al. 2020, 2022). DRL has also been applied to complex supply chain structures, including multi-echelon systems (Wu et al. 2023, Alvo et al. 2023, Liu et al. 2024, Stranieri et al. 2024), one-warehouse multi-retailer networks (Kaynov et al. 2024), and the stochastic capacitated lot-sizing problem (Van Hezewijk et al. 2023).

Despite these algorithmic advances, a deployment gap remains. These methods require formal mathematical modeling and hyperparameter tuning that are inaccessible to the average small-business manager. Our work does not seek to invent a new DRL algorithm; rather, we leverage these existing powerful solvers (the “Optimization Agent”) and focus on the *interface* required to make them usable by non-experts.

1.2.2. LLMs for Optimization and Decision Support. The emergence of LLMs has sparked intense interest in automating decision making (e.g. Huang et al. (2025)). LLMs are increasingly being adopted as versatile decision-support tools across many critical sectors. Within business

and operations management, LLMs are leveraged to optimize processes, improve efficiency, and drive innovation (Li et al. 2024). Significant recent research interest has focused on their ability to automatically formulate optimization problems and dynamic programming problems from natural language descriptions (AhmadiTeshnizi et al. 2023, 2024, Zhou et al. 2025, Liang et al. 2025). In supply chain management specifically, LLMs are used for tasks such as demand forecasting and logistics optimization (Lu et al. 2024), and recent work has also focused on developing agentic frameworks for advanced applications (Quan and Liu 2024, Qi et al. 2025, Long et al. 2025, Simchi-Levi et al. 2026, Cohen et al. 2025). In the healthcare domain, LLMs show remarkable potential to augment clinical decision-making by rapidly synthesizing patient information, generating differential diagnoses, and suggesting treatment plans, positioning them as powerful co-pilots for clinicians (Shah et al. 2023, Nazi and Peng 2024). Similarly, the financial sector uses LLMs for a wide range of applications, from market analysis to risk management, with specialized models like FinGPT and BloombergGPT enabling nuanced sentiment analysis, automated report generation, and enhanced algorithmic trading strategies.

Our work advances this direction by addressing the *ambiguity* of the inputs. Existing frameworks typically assume the user provides a complete problem description. In contrast, our *Information Extraction Agent* assumes the user is boundedly rational and the problem is initially ill-posed, requiring an iterative dialogue to elicit necessary parameters (e.g., distinguishing backorders from lost sales) before the solver can be invoked. This design makes our framework particularly suitable for small and medium-sized enterprises that lack in-house analytical staff but still face nontrivial inventory trade-offs.

1.2.3. Generative Agents and User Simulation. Evaluating interactive decision support systems presents a methodological dilemma: static datasets (e.g., standard NLP benchmarks) fail to capture the multi-turn dynamics of problem formulation, while human-subject trials are resource-intensive and difficult to reproduce. To address this, we draw upon the rich history of user simulation in dialogue systems and the recent emergence of generative agents.

User simulators have long been a staple in training task-oriented dialogue systems, particularly for Reinforcement Learning (RL) agents. Early approaches relied on agenda-based mechanisms (Schatzmann et al. 2007), where the simulator followed a strict stack of goals (e.g., “book a flight”, “specify time”). While effective for slot-filling tasks, these rule-based systems lacked the linguistic diversity of real users. The field subsequently moved toward data-driven approaches, utilizing sequence-to-sequence models to learn user behavior directly from corpora (Kreyssig et al. 2018, Shi et al. 2019). However, these models often struggled with response collapse, generating generic or repetitive answers that failed to challenge the system. The advent of Large Language Models has

revolutionized user simulation by enabling “Generative Agents” that maintain consistent personas and memories. Park et al. (2023b) shows that LLMs could simulate believable social interactions in a sandbox environment. In the context of task-oriented dialogue, Wang et al. (2025) established that LLMs can act as zero-shot user simulators that outperform traditional models. Crucially for our work, recent literature emphasizes simulating human imperfections. Wang et al. (2024) investigated role-playing capabilities, finding that LLMs can effectively mimic specific demographic traits and knowledge gaps. Li et al. (2023) introduced “CAMEL”, a framework where two LLM agents (a user and an assistant) interact autonomously to solve tasks, revealing that agent-to-agent simulation can uncover edge cases that static testing misses.

Our *Human Imitator* synthesizes these streams. Unlike the perfect “agenda-based” simulators of the past, our agent is designed to replicate the “bounded rationality” (Simon 1955) of a non-expert supply chain manager. By conditioning the simulator on personas that exhibit specific knowledge gaps (e.g., confusing “lost-sale” with “back-order”), we create a rigorous testbed that evaluates not just the system’s ability to solve math, but its ability to clarify ambiguity. By fine-tuning a model on real human-machine interactions, we create a “digital twin” of a small business owner. This allows us to stress-test the system’s ability to handle noise and ambiguity at scale, providing a more rigorous evaluation metric than simple code-generation accuracy.

2. The Hybrid Agentic Framework

In this section, we present the design of our LLM-based solver. The core philosophy of our framework is the strict separation of concerns: rather than employing a single “black box” model, we architect a modular system where specialized agents handle distinct cognitive tasks: ambiguity resolution, mathematical optimization, and semantic interpretation.

This design enables us to bridge the gap between the ambiguous descriptions often found in real-world business and the precise inputs required by formal inventory models (formulated in Section A). By organizing these agents into a collaborative workflow, our system functions as a “cognitive scaffold”: it standardizes human inputs for rigorous solving and subsequently interprets the mathematical results into actionable insights.

We begin by describing the structure of the agentic pipeline, followed by a detailed breakdown of each individual agent.

2.1. Pipeline Architecture

Our framework integrates human-like problem input, structured information extraction, adaptive optimization, and verbal policy interpretation into a coherent pipeline (Figure 1). The pipeline is composed of three interconnected components that together transform informal descriptions into optimized inventory policies.

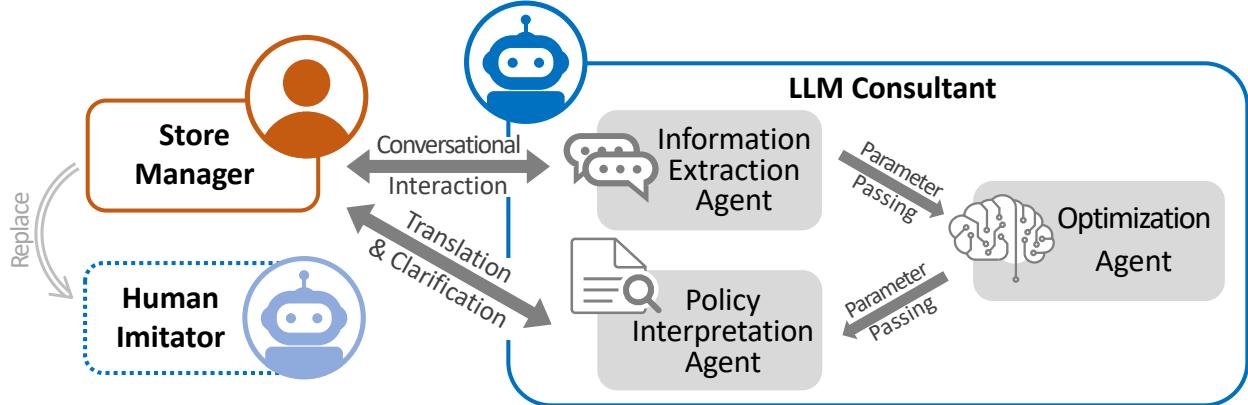


Figure 1 LLM-Agent Pipeline for Inventory Management

Information Extraction Agent. This agent converses with the human user, interprets the user’s informal input, and incrementally converts it into structured parameters. Through conversational rounds, it asks clarifying questions, fills missing entries, and resolves conflicts until a complete and consistent parameter table is obtained.

Optimization Agent. Once the problem is fully specified, the Optimization Agent invokes appropriate solvers, such as those for the classical (s, S) policy and deep reinforcement learning algorithms, to generate inventory policies. The choice of the recommended policy depends on user-defined preferences, e.g., trading off expected cost against cost variance.

Policy Interpretation Agent. Once the Optimization Agent identifies a recommended policy, this policy is passed to the Policy Interpretation Agent, which intuitively explains the nature of the policy to the user, responds to optimal order quantity queries, and addresses any clarification questions.

Together, these components establish an iterative, human-like workflow that mimics real managerial practice while remaining fully automatable.

2.2. Information Extraction: Turning Dialogue into Parameters

When a user approaches our system for help with an inventory problem, the primary barrier to applying operations research methods is not optimization itself, but *problem specification*: converting an informal, natural-language business description into a fully specified mathematical model. In our setting, the necessary information is provided incrementally through a multi-turn conversation, so the system must continuously extract, reconcile, and refine model parameters as the dialogue evolves. We assign this responsibility to the **Information Extraction Agent** (built on *GPT-5-mini*¹).

¹ Unless otherwise noted, *GPT-4o* serves as the baseline LLM throughout this paper. *GPT-5-mini* is a smaller and weaker model; we use it here to reduce cost, as its performance is sufficient for the information-extraction task.

In such conversational AI systems, relying solely on the model’s implicit memory—that is, the context window—is brittle for operations research applications. Long, unstructured dialogues are prone to *context drift*, in which previously specified constraints are forgotten, and *hallucination*, in which missing parameters are spuriously fabricated. To mitigate these failure modes, we adopt an *Explicit Memory* architecture. Concretely, the agent is required to maintain a persistent *Parameter Specification Table* (e.g., Table 1) that represents the system’s authoritative epistemic state.

The agent’s objective is then to populate this table by systematically eliciting, inferring, and validating all decision-relevant primitives. Through this process, the agent transforms an initially empty template (e.g., Table 1) into a fully instantiated problem representation (e.g., Table 2) that can be passed directly to the solver.

Parameter	Value	Unit	Status
time_horizon	—	—	undefined
demand_type	—	N/A	undefined
demand_distribution	—	N/A	undefined
perishability	—	N/A	undefined
state_transition_model	—	N/A	undefined
holding_cost	—	—	undefined
penalty_cost	—	—	undefined
setup_cost	—	—	undefined
lead_time	—	—	undefined
max_inventory	—	N/A	undefined
max_order	—	N/A	undefined
risk_tolerance	—	N/A	undefined

Table 1 Initial Parameter Specification for the Information Extraction Agent

Parameter	Value	Unit	Status
time_horizon	60	days	defined
demand_type	random	N/A	defined
demand_distribution	Poisson($\lambda = 10$)	N/A	defined
perishability	TRUE	N/A	defined
state_transition_model	lost_sale	N/A	defined
holding_cost	5	USD/unit/day	defined
penalty_cost	15	USD/unit	defined
setup_cost	100	USD/order	defined
lead_time	5	days	defined
max_inventory	100	N/A	defined
max_order	20	N/A	defined
risk_tolerance	3	N/A	defined

Table 2 Example of a Completed Parameter Specification

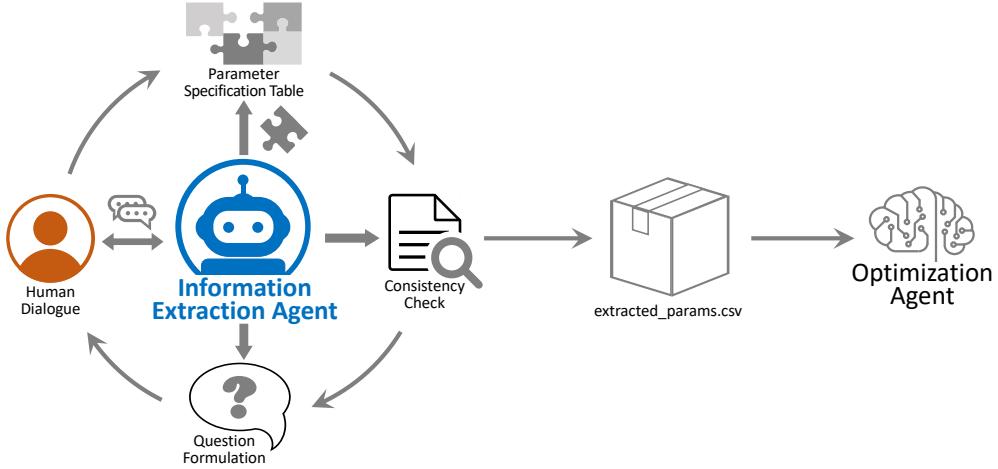


Figure 2 Architecture of the Information Extraction Agent

More concretely, the conversational process unfolds in rounds. In each round, the Information Extraction Agent analyzes the user’s latest response and performs the following actions:

1. *Update Table*: If new, credible information is identified, it fills the corresponding blank in the table and updates the status to “defined.”
2. *Detect Conflicts*: If the user provides information that contradicts a previously defined entry, the agent flags the conflict for resolution.
3. *Formulate Question*: If the table contains undefined variables or identified conflicts, the agent generates a targeted question to clarify ambiguities or gather missing data. The conversation continues to the next round.

The dialogue concludes when all parameters are marked as “defined” and the system verifies that no conflicts exist. A visual representation of this complete workflow is provided in Figure 2. For reproducibility, the exact system prompt used by the Information Extraction Agent is documented in Table 8 (Section C.1). Furthermore, we provide representative conversation logs in Sections C.2 and 4.3; specifically, Examples 3–5 highlight the agent’s capability to navigate ambiguity and resolve logical conflicts effectively.

Upon successful termination, the Information Extraction Agent saves two files:

- `extracted_params.csv`, containing the extracted parameters from the conversation; and
- `conversation_log.txt`, containing the complete conversation history.

Finally, it invokes the Optimization Agent, which uses the `extracted_params.csv` file to calculate and propose a recommended inventory policy.

REMARK 1 (COMPARISON WITH DPLM). With the above extraction-and-memory mechanism in place, we now situate our approach relative to recent efforts on automating operations-research modeling. We position our work by contrasting it with the recent Dynamic Programming Language

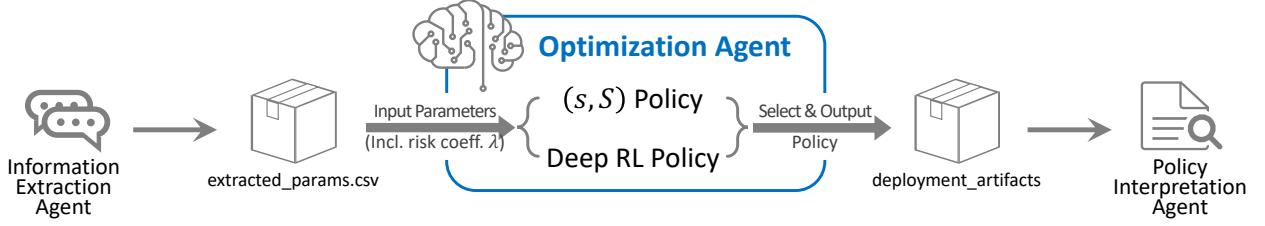


Figure 3 Architecture of the Optimization Agent

Model (DPLM) proposed by Zhou et al. (2025). While sharing the ultimate goal of automating operations research modeling, our approaches differ fundamentally in their handling of information ambiguity and training methodology. Zhou et al. (2025) demonstrate that a specialized fine-tuning pipeline trained on synthetic data can effectively translate fully specified, static textbook descriptions into mathematical models. In contrast, our framework addresses the *preceding* challenge of real-world ambiguity, where problem definitions must be interactively elicited and maintained through unstructured dialogue. We demonstrate that for this extraction phase, rigorous prompt engineering with a capability-rich foundation model (*GPT-5-mini*) and explicit memory management yields high-fidelity results without requiring the extensive supervised fine-tuning and reinforcement learning resources detailed in Zhou et al. (2025). However, their rigorous synthetic data generation (DualReflect) and fine-tuning recipes offer a valuable roadmap for future enhancements, particularly for distilling our extraction capabilities into smaller, more efficient models or for improving the system’s ability to auto-formulate novel constraints that fall outside standard inventory templates.

2.3. Optimization: From Parameters to Policies

With the problem structure specified, the **Optimization Agent** then executes the core algorithmic work. Unlike a generic LLM that might “hallucinate” a solution, this agent acts as a dispatcher for rigorous operations research solvers. The corresponding workflow is illustrated in Figure 3.

A key feature of our optimization module is the integration of user risk preferences. We recognize that normative optimality is not single-dimensional; users often trade off expected cost against stability. To capture this, the agent elicits a preference parameter $\lambda \in [-10, 10]$, which weights the penalty for variance (risk). The agent then selects the policy π that minimizes the following composite objective:

$$\mathcal{L}(\pi) = \mathbb{E}[\text{Cost}(\pi)] + \exp(-\lambda) \cdot \text{Std}[\text{Cost}(\pi)], \quad (1)$$

where $\text{Cost}(\pi)$ represents the expected cost and $\text{Std}(\pi)$ represents the cost standard deviation under policy π .

In our current implementation, the agent arbitrates between two distinct policy classes, representing the spectrum from classical theory to modern AI:

The (s, S) Policy (Classical Heuristic): The (s, S) policy is a classic inventory management approach defined by a reorder point s and an order-up-to level S . When the inventory level (the sum of on-hand inventory and orders in the pipeline) drops to or below s , a replenishment order is triggered to bring the stock back up to S . This policy effectively balances ordering and holding costs by maintaining inventory within these thresholds, ensuring stock is replenished only when necessary. It is particularly effective in systems with positive lead times. While the policy is simple to implement and interpret, improper values for s and S can lead to significant overstocking or stockouts. Our Optimization Agent simulates the expected cost and variance across a grid of (s, S) parameter choices to identify the best (s, S) configuration.

Deep Q-Network (Deep Reinforcement Learning): The DQN policy is based on the Markov Decision Process (MDP) formulation of the inventory problem (detailed in Section A). It is a reinforcement learning method that combines Q-learning with deep neural networks to learn optimal policies in dynamic environments. Rather than maintaining a Q-table, which is infeasible for large state-action spaces, DQN uses a neural network to approximate the Q-value function, predicting the expected cumulative reward for each action given a state. We define the state space as the composition of the current inventory position, the time period, and the vector of orders in the pipeline. By iteratively updating the Q-network via the Bellman equation, DQN approximates optimal policies using samples from the demand distribution. Notably, DQN performs well even when the lead time is large due to its ability to handle sequential decision making problems with high-dimensional input.

These two paradigms represent an inherent trade-off between *expected efficiency* and *operational stability*. While the Deep RL policy leverages high-dimensional feature extraction to aggressively minimize expected costs, its stochastic nature may introduce higher variance. Conversely, the (s, S) heuristic, though potentially conservative, offers a robust and predictable baseline. Our framework addresses this dichotomy through an *agentic design*: the Optimization Agent interprets the user's risk tolerance and performs meta-reasoning to arbitrate between the solvers, selecting a policy that balances expected cost and operational stability.

Upon convergence, the Optimization Agent serializes the model artifacts to a directory named `deployment_artifacts`, containing three files:

- `config.json`: Includes important instance parameters (e.g., lead time, maximum inventory, time horizon, and the state and action dimensions of the DQN network).
- `dqn_weights.pth`: Contains the weights of the DQN network.
- `policy_evaluation_results.csv`: Stores the average cost and standard deviation of the optimal DQN policy and (s, S) policy, respectively, along with the optimal s and S values.

These artifacts serve as the essential input for the subsequent Policy Interpretation Agent.

REMARK 2 (EXTENSIBILITY TO NOVEL ALGORITHMS). Although our experiments focus on (s, S) and DQN policies, the modular architecture of the Optimization Agent facilitates the seamless integration of additional policy classes and emerging algorithms. Unlike black-box AI models, where enhancing algorithmic reasoning often necessitates resource-intensive retraining or fine-tuning, our framework treats solvers as interchangeable modules. This design allows researchers to effortlessly incorporate novel optimization techniques, thereby ensuring the system remains at the frontier of operations research without altering the core linguistic interface.

2.4. Policy Interpretation: Communicating Policies in Words

The final component, the **Policy Interpretation Agent**, ingests the artifacts produced by the Optimization Agent and translates them into actionable business intelligence. This stage is critical for realizing actual cost reductions: theoretical optimality translates to practical efficiency only when human decision-makers trust and adhere to the algorithmic suggestions.² To mitigate algorithm aversion and foster cognitive alignment, this agent navigates the trade-off between *performance* (lower cost $\mathcal{L}(\pi)$ in eq. (1)) and *interpretability*.

Because the (s, S) policy is generally more interpretable and familiar to practitioners, while the DQN policy is less transparent but typically achieves lower costs, the Decision Interpretation Agent follows the communication protocol below:

Scenario A: The (s, S) Policy Performs Better. If the (s, S) heuristic yields a lower objective value $\mathcal{L}(\pi)$, the agent leverages its inherent interpretability. It provides a detailed yet accessible explanation of the reorder point s and order-up-to level S , describing how the policy operates in day-to-day inventory decisions. It then invites the user to ask follow-up questions or relate the thresholds to their own operational experience.

Scenario B: The DQN Policy Performs Better. If the DQN policy yields a lower objective value $\mathcal{L}(\pi)$, the agent first offers a high-level overview of the black-box policy, framing it as an “on-demand expert”: at any time step, given the current inventory position and pipeline orders, the user can query the agent for a recommended order quantity. If the user finds this lack of interpretability unsatisfying and prefers a more intuitive explanation, the agent then additionally presents the suboptimal (s, S) policy, reports the percentage difference in the optimization objective between the two policies (e.g., “If you’re interested in a simpler policy, the $(s = 2, S = 11)$ policy exists but it costs 20.5% more”), and allows the user to decide which policy they ultimately wish to adopt.

² Pre-LLM studies of automated replenishment already document that store managers frequently deviate from optimization-based order recommendations and that accounting for their behavioral responses is essential for effective system design ([Van Donselaar et al. 2010](#)).

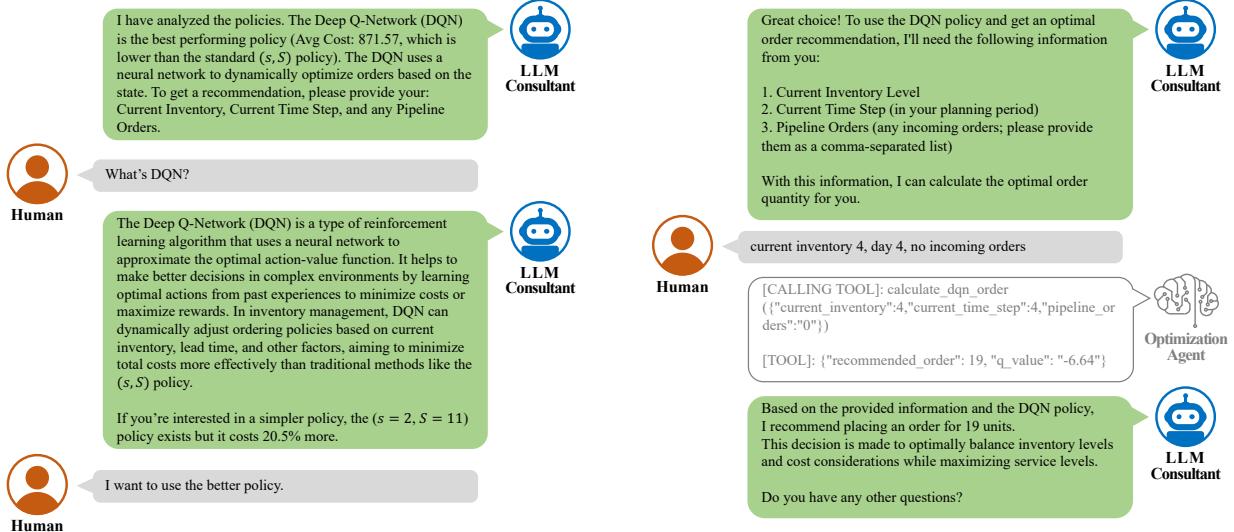


Figure 4 Representative dialogue between the Human User and the Policy Interpretation Agent.

Throughout this interaction, the agent remains responsive to user inquiries, clarifying the rationale behind specific recommendations to reinforce trust. Once the user indicates understanding or signals readiness to adopt the policy, the agent terminates the conversation and archives the dialogue transcript as `interpret.txt` for audit and analysis. An illustrative example of this interaction is provided in Figure 4.

3. The Human Imitator: A Scalable Testbed

To rigorously evaluate our agentic framework presented in Section 2, we must confront a fundamental methodological challenge: the scarcity of realistic testbeds. Standard inventory benchmarks rely on static, structured parameters, failing to capture the ambiguity, inconsistency, and bounded rationality that define real-world decision-making. Conversely, human-subject experiments, while ecologically valid, are costly, unscalable, and difficult to reproduce.

We bridge this gap by developing the *Human Imitator*—an LLM fine-tuned to function as a “digital twin” of a retail manager. Unlike a standard LLM, which strives for helpfulness and precision, our imitator is explicitly trained to reproduce the idiosyncrasies of human behavior: providing rough estimates, using informal language, and occasionally omitting critical details. This allows us to stress-test our system against realistic “human noise” at scale.

3.1. Data Collection: Capturing Managerial Dialogue

We constructed a specialized corpus of inventory management dialogues through a controlled study. We deployed an online interface where participants played the role of a store manager interacting with a generic “Base Model” consultant. The Base Model was prompted to elicit specific operational

parameters—including demand patterns, perishability, holding costs, and lead times—while the participants were instructed to describe their business scenario using natural language.

This process yielded a dataset of prompt–response pairs, $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$, where $x^{(i)}$ represents the system’s inquiry (e.g., “*How long do items stay fresh?*”) and $y^{(i)}$ represents the human’s verbatim response (e.g., “*maybe 3 days max*”).

In total, 66 participants generated $N = 1,184$ high-quality conversational turns. This dataset captures a diverse range of linguistic styles, from precise specifications to vague, heuristic-driven descriptions. Detailed screenshots of the interface and example transcripts of the collected dialogues are provided in Section B.

3.2. Supervised Fine-Tuning (SFT)

To instill human-like behavioral patterns into the model, we performed Supervised Fine-Tuning (SFT) on the *Qwen2.5-7B* foundation model. We treat the imitator as a policy π_θ parameterized by weights θ . The training objective is to maximize the likelihood of generating the empirically observed human response y given the context x :

$$\max_{\theta} \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} \log \pi_\theta(y^{(i)} | x^{(i)}).$$

This objective forces the model to align its probability distribution with human behavior, effectively teaching it to “unlearn” robotic precision and adopt the persona of a boundedly rational manager.

Since our goal is *imitation*—creating a faithful proxy that reproduces the specific distribution of our target population—we train on the full dataset to maximize behavioral coverage. Training was conducted using Low-Rank Adaptation (LoRA) on a single NVIDIA L4 GPU (24GB). The optimization converged rapidly over 3 epochs (524 minutes), with the loss function reducing from 1.9234 to 0.3473, indicating that the foundation model successfully adapted to the domain-specific linguistic patterns. See Section B.2 for experimental details.

3.3. Empirical Validation: Quantifying Behavioral Alignment

Does the fine-tuned model truly resemble a human manager? We monitor the model’s fidelity across four metrics that measure alignment at distinct linguistic levels: *fluency*, *structure*, *token usage*, and *sentence meaning*. The training trajectory is visualized in Figure 5, with numerical results detailed in Table 3.

Drastic Reduction in Perplexity (Domain Adaptation): The most critical indicator of success is Perplexity (PPL), which measures the model’s predictive uncertainty regarding the next token. As shown in Table 3, the PPL drops precipitously from 37.99 (Base Model) to 3.33 (Epoch 3). This order-of-magnitude improvement confirms that the model, which initially spoke like a generic assistant, has successfully adopted the specific jargon, hesitation, and vernacular of human input.

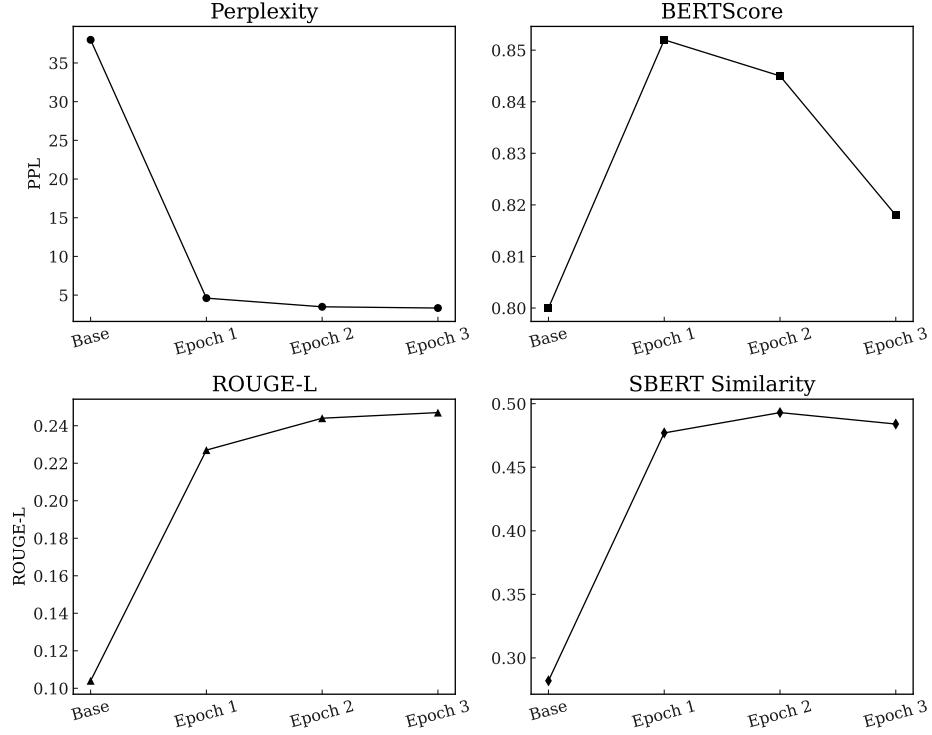


Figure 5 SFT Evaluation Trajectory. The curves illustrate the rapid convergence of the Human Imitator. The drop in Perplexity and the simultaneous rise in semantic similarity metrics (BERTScore, ROUGE-L and SBERT) indicate successful domain adaptation.

Training Stage	PPL↓	BERTScore↑	ROUGE-L↑	SBERT↑
Epoch 0 (Base)	37.991	0.800	0.104	0.282
Epoch 1	4.615	0.852	0.227	0.477
Epoch 2	3.487	0.845	0.244	0.493
Epoch 3	3.334	0.818	0.247	0.484

Table 3 Validation of the Human Imitator. We track four metrics across training epochs. PPL measures predictive certainty (lower is better). BERTScore measures token-level embedding similarity. ROUGE-L measures structural Longest Common Subsequence (LCS) overlap. SBERT measures sentence-level semantic alignment.

Semantic and Structural Convergence: To ensure the model captures the *intent* and *style* of the managers, we examine similarity metrics at three levels of abstraction:

- *BERTScore (Token-Level Embedding Similarity):* BERTScore computes the cosine similarity between contextual embeddings of individual tokens. A high score (e.g. 0.85) confirms that the model is selecting the correct *vocabulary* within context, ensuring granular alignment with the specific terminology used by inventory managers.
- *ROUGE-L (Structural Similarity):* ROUGE-L measures the Longest Common Subsequence (LCS) between the generated and reference text. The significant rise (from 0.104 to 0.247)

indicates that the imitator is replicating the *syntactic structure* of the participants—mimicking the sentence fragments, brevity, and informal phrasing typical of the domain.

- *SBERT (Sentence-Level Semantic Similarity):* SBERT computes embeddings for the entire sentence. The doubling of this metric (from 0.282 to 0.493) confirms that the model is capturing the *holistic semantic intent* of the human response, ensuring that the generated “noise” preserves the underlying meaning of the parameter description.

Based on these results, we selected the checkpoint at **Epoch 2** for our experiments. While Epoch 3 achieves a marginally lower perplexity, we observe a degradation in semantic alignment, evidenced by the sharp drop in BERTScore (from 0.845 to 0.818). In contrast, Epoch 2 achieves the highest SBERT similarity (0.493) while maintaining a high BERTScore. This suggests that Epoch 2 offers the optimal trade-off: it preserves the highest semantic fidelity to human intent without overfitting to the specific lexical patterns of the training data. The result is a robust, scalable proxy that authentically simulates the “last mile” friction of inventory management.

4. Evaluation

In this section, we transition from theoretical design to empirical validation. Our evaluation is designed to measure the extent to which our hybrid agentic framework bridges the gap between *descriptive reality* (ambiguous human inputs) and *normative optimality* (mathematical best practices). Specifically, we aim to quantify the efficiency loss incurred when relying on the approximate reasoning of general-purpose LLMs rather than explicit stochastic optimization. By carefully isolating sources of error through controlled baselines, we seek to understand not only *if* the system works, but also *why* standard foundation models fail in this domain.

We address three primary research questions:

1. *Efficacy:* Does our hybrid architecture, by decoupling semantic understanding from logical optimization, outperform advanced commercial models (e.g., *GPT-4o*) on complex stochastic inventory control tasks?
2. *Structural Robustness:* Are the performance gains stable across varying degrees of environmental complexity? Under what conditions does our agentic framework achieve the largest marginal improvements?
3. *Limits of Prompt-Based Reasoning:* Can the reasoning failures of general-purpose LLMs be remedied simply by providing perfect, ground-truth problem specifications? Or does stochastic optimization mark a fundamental *computational boundary* that probabilistic language models cannot reliably cross without additional algorithmic and architectural support?

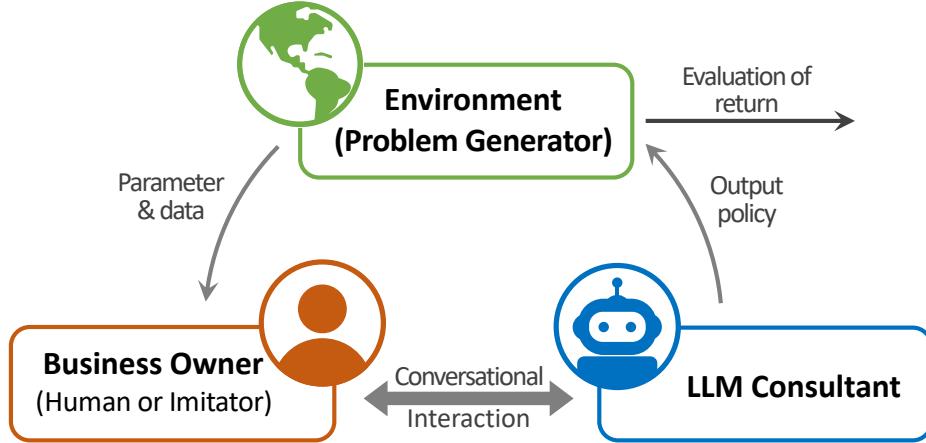


Figure 6 Evaluation Protocol. The workflow moves from latent ground-truth generation to conversational extraction, culminating in objective policy assessment.

4.1. Evaluation Pipeline: An *In Silico* Laboratory

To empirically validate the Hybrid Agentic Framework proposed in Section 2, we construct a controlled *in silico* experiment using the Human Imitator developed in Section 3. This experimental design allows us to stress-test the system under realistic conversational friction, including ambiguity, inconsistency, and non-technical phrasing, while avoiding the logistical and financial constraints of large-scale human-subject trials.

Notably, this evaluation protocol generalizes beyond inventory management. By employing a high-fidelity synthetic agent to simulate the human-in-the-loop, our approach provides a scalable and reproducible testbed for evaluating a wide range of interactive LLM tasks where human variability and latent business knowledge are critical factors.

4.1.1. Experimental Protocol. The evaluation protocol for each trial follows a three-stage lifecycle, as illustrated in Figure 6:

1. *Initialization (Ground Truth Generation):* The process begins with a Problem Generator Agent, implemented using *GPT-4o*, which procedurally generates unique inventory management instances. Each instance comprises a high-level semantic context (e.g., “Managing imported ingredient inventory for a premium restaurant”) and a precise set of *ground-truth parameters*³ (Table 4). These parameters constitute the “latent reality” of the business, representing details that a manager might intuitively grasp or access via records, yet struggles to articulate formally due to partial awareness or bounded rationality. Crucially, these ground-truth

³ When generating parameters, we follow standard parameterizations in the inventory management literature. For example, the inventory holding cost is set to 30% of the item cost; the standard deviation of demand is approximately one-third of the mean; and the *critical ratio*, defined as `penalty_cost / (penalty_cost + holding_cost)`, typically lies between 0.8 and 0.9.

values provide the definitive benchmark against which all policies subsequently generated by the systems are evaluated.

Parameter	Description
<code>time_horizon</code>	The planning period for inventory decisions (default unit: days).
<code>demand_type</code>	The nature of customer demand, categorized as either <i>deterministic</i> or <i>random</i> .
<code>demand_distribution</code>	If deterministic, a constant value; if random, a statistical distribution (e.g., $\text{Normal}(\mu = 30, \sigma = 10)$). <i>(While our experiments primarily utilize Normal or Poisson distributions for simplicity, the framework supports a “Custom” setting that allows the system to infer distributions directly from user-uploaded historical data.)</i>
<code>perishability</code>	A boolean value indicating if the items expire.
<code>state_transition_model</code>	The outcome of a stockout, modeled as either a <i>lost sale</i> or a <i>backlog</i> .
<code>holding_cost</code>	The cost to store one unit of inventory for a specific time period.
<code>penalty_cost</code>	The cost incurred for each unit of unmet demand.
<code>setup_cost</code>	The fixed cost associated with placing an order.
<code>lead_time</code>	The time delay between placing an order and receiving it.
<code>max_inventory</code>	The maximum allowable inventory level.
<code>max_order</code>	The maximum quantity that can be ordered at one time.
<code>risk_tolerance</code>	An integer λ in $\{-10, -9, -8, \dots, 9, 10\}$, where a lower value indicates higher risk aversion.

Table 4 Inventory Environment Parameters. These variables define the ground truth against which all policies are evaluated.

2. *Interaction:* We initialize the Human Imitator with the latent reality provided by the generator.

It then engages a target system (selected from the treatment groups in Section 4.1.2) in a natural language dialogue. Crucially, the Imitator simulates the opacity of real-world consulting: rather than transmitting structured parameters directly, it describes the business situation conversationally, revealing information only through narrative descriptions and responses to follow-up inquiries, while maintaining realistic levels of ambiguity.

3. *Assessment (Policy Evaluation):* Once the system proposes a policy, we evaluate its performance against the original *ground-truth* parameters (distinct from the parameters the system may have inferred). We estimate the expected cost and standard deviation via Monte Carlo simulation. To accommodate different output formats, we employ a unified evaluation protocol:

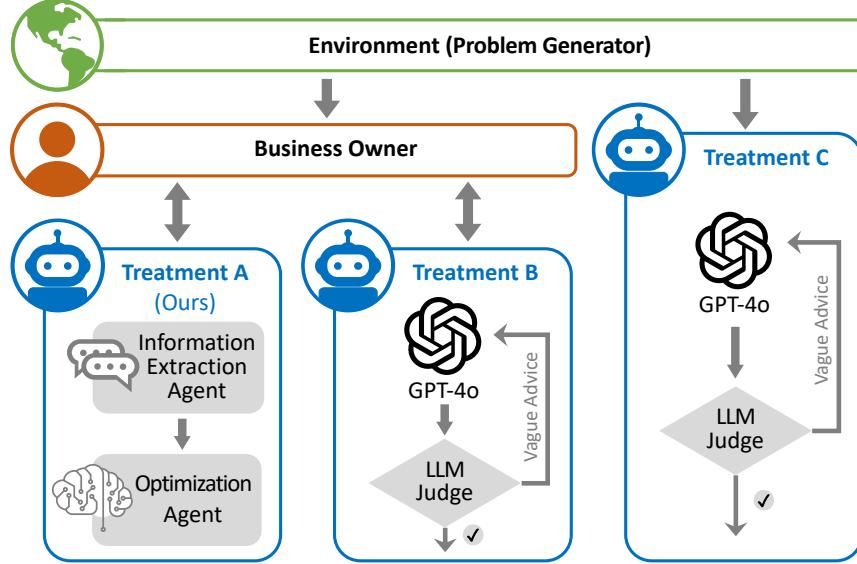


Figure 7 Overview of the comparative experimental framework.

- *Automated Evaluation:* For standard mathematical policies (specifically Deep RL, (s, S) , or (R, Q) ⁴), our system automatically parses the parameters and executes the simulation.
- *Manual Translation:* For policies proposed by baseline methods that do not fit these categories, we manually translate the described logic into executable code to ensure they are evaluated fairly.

Finally, we compute the optimality gap between the proposed policy and the theoretical benchmark.

This procedure is repeated across multiple trials to generate diverse inventory problem instances. Given that optimal costs vary significantly in magnitude across different scenarios, we report the relative cost difference between algorithms.

4.1.2. Experimental Conditions (Baselines). To isolate the sources of performance gain, we compare our proposed framework (in Section 2) against two distinct *GPT-4o* baselines. This comparative design allows us to decompose decision error into *epistemic error* (translation failure) and *computational error* (optimization failure). Figure 7 provides a schematic overview of this experimental architecture, mapping the information flow from the latent ground truth (Environment) to the final mathematical policy (output) for each treatment.

Treatment A: Interactive Hybrid Agent (Ours). This is the full proposed architecture in Section 2. The system must navigate the conversation with the Human Imitator to extract parameters into a structured format (Table 1) and then employ its internal Optimization Agent to

⁴ The (R, Q) policy is an inventory control strategy where a fixed quantity Q is ordered whenever the inventory position drops to or below the reorder point R .

compute a normative policy, using either a grid search for optimal (s, S) thresholds or Deep Reinforcement Learning to handle complex, high-dimensional state spaces.

Treatment B: GPT-4o (Interactive). This baseline represents the standard “End-to-End” LLM approach. To ensure a rigorous and fair comparison, the Human Imitator initiates the conversation with the identical starting context used for Treatment A, the Hybrid Agentic System. Operating without access to external algorithmic solvers, the model must simultaneously navigate the dialogue context, infer the underlying operational parameters, and derive a policy based solely on its internal probabilistic weights. This treatment specifically tests the capacity of general-purpose models to overcome epistemic barriers and perform complex stochastic reasoning within a purely natural language interface.

Treatment C: GPT-4o (Parameter Input). This non-interactive baseline assesses the model’s intrinsic inventory optimization capabilities under conditions of *perfect information*. To bypass the ambiguity associated with conversational extraction, we directly feed the *ground-truth parameter file*, a structured representation of the latent reality, to *GPT-4o*, rather than relying on narrative extraction. The model is then explicitly prompted to derive an optimal policy. This counterfactual setup isolates computational reasoning from linguistic interpretation, allowing us to investigate whether the performance bottleneck in interactive settings stems from dialogue opacity or the model’s fundamental algorithmic limitations.

Control Measures and Prompting. To ensure fairness and consistency, both *GPT-4o* baselines (Treatments B & C) are subject to strict output controls. Specifically, they are prompted with the following instruction to ensure the generation of *evaluable policies* that are easily translatable into mathematical formulas, rather than vague qualitative advice:

The final recommendation must have clear numbers, specify actions for all possible scenarios, and be easily translatable into a mathematical formula.
 For example, “When inventory falls below 20 units, order 80 units” is a strategy.
 “I recommend an (s, S) policy: when the sum of on-hand and on-order inventory drops below 35 units, order up to a total of 150 units” is also a clear strategy.
 “We should consider optimizing inventory” or “You need a better plan” are not strategies.

Without this instruction, *GPT-4o* often generates underspecified recommendations—such as “you need to order more when inventory is below 5 units”—which cannot be explicitly deployed or analyzed in simulation.

To operationalize these constraints, we further employ an auxiliary *LLM Judge* (built on *GPT-4o*). This judge monitors the response stream and acts as a termination mechanism: it validates whether the output contains a well-defined numerical policy and, upon confirmation, halts the generation to archive both the conversation log and the extracted policy for simulation.

4.2. Experimental Results and Findings

We report our findings based on a rigorous evaluation across 70 randomly generated inventory scenarios. To ensure a standardized comparison, we evaluate performance based on expected cost minimization ($\mathbb{E}[\text{Cost}(\pi)]$), assuming a risk-neutral decision-maker (i.e. $\lambda = 10$ in eq. (1)).

Table 5 summarizes the comparative performance. The results provide strong empirical support for our architectural thesis: separating semantic understanding from logical optimization significantly outperforms end-to-end LLM reasoning.

Benchmark & Scenario	Count (n)	Avg. Cost Red. (Mean & Std. Dev.)	Win Rate (%)
Baseline 1: vs. GPT-4o (Interactive)			
Overall Performance	70	32.1% \pm 17.8%	97.1%
<i>By Demand Distribution (Lead Time = 7 days)</i>			
Normal Distribution	14	34.2% \pm 15.0%	100%
Poisson Distribution	11	33.8% \pm 19.7%	100%
<i>By Lead Time (Demand = Poisson)</i>			
Short Lead Time (1 day)	15	26.8% \pm 16.4%	93.3%
Long Lead Time (7 days)	11	33.8% \pm 19.7%	100%
Baseline 2: vs. GPT-4o (Parameter Input)			
Overall Performance	70	33.4% \pm 16.8%	100%

Table 5 Performance Statistics of Hybrid Agentic Framework.

This table reports the cost reduction statistics (Mean and Standard Deviation) and the win rate (percentage of instances where the Hybrid Agent outperformed the benchmark).

To contextualize the economic scale of these experiments, our simulation parameters mirror the operating realities of Small and Medium-sized Enterprises (SMEs). The holding cost in our testbed ranges from \$0.25 to \$4.00 per unit/period, with penalty costs for stockouts ranging significantly higher (averaging roughly \$5.00). Over a standard planning horizon of a few months, the cumulative cost for a single item typically runs into the thousands of dollars. While these unit economics may appear modest in isolation, they scale linearly across the hundreds of SKUs typical of a retail inventory. Consequently, a persistent efficiency gap of roughly 30% translates into substantial margin erosion for a real-world business, validating the material significance of our findings.

In the following subsections, we unpack the sources of this advantage in three steps. First, we quantify the aggregate efficiency gap—the “Hallucination Tax”—in interactive settings (Section 4.2.1). Second, we identify the operational conditions that make the agent most valuable (Section 4.2.2). Finally, we isolate the root cause of the baseline’s failure by distinguishing between errors of understanding (bad inputs) and errors of reasoning (bad logic) (Section 4.2.3).

4.2.1. The Hallucination Tax (*Treatment A* vs. *B*). Our comparison begins with the aggregate performance gap between our Hybrid Agentic Framework (*Treatment A*) and the *GPT-4o* Interactive baseline (*Treatment B*). As detailed in Table 5, the hybrid approach yields a substantial **32.1%** cost reduction (standard deviation 17.8%) and achieves a 97.1% win rate across the 70 test instances. In the rare exceptions (2 instances) where *Treatment B* performs better, the margins are negligible (2% and 6%, respectively), and in both cases, the baseline happens to converge to an (s, S) heuristic effectively identical to the one identified by our optimizer.

We term this efficiency loss the “**Hallucination Tax**.” It represents the compound penalty incurred when a firm relies on a general-purpose language model as an end-to-end solver. This gap is not merely a reflection of random error, but a structural divergence in how the two systems approach problem-solving: one relies on probabilistic token generation, the other on rigorous mathematical optimization.

The Fluency Trap: Why Managers Get Fooled. A qualitative analysis of the conversation logs reveals a dangerous phenomenon we call the *fluency trap*. The *GPT-4o* baseline is highly eloquent; it correctly uses domain terminology and offers structurally plausible advice. However, this linguistic competence masks a fundamental inability to calibrate policy parameters to specific constraints.

A representative example is observed in the dialogue in Figure 9. Despite explicitly acknowledging a maximum inventory capacity of 80 units, the *GPT-4o* model confidently recommends: “*When your total inventory... drops to or below 89 units, place an order to bring the inventory up to 80 units.*” This recommendation is not merely suboptimal; it is physically impossible.

For a manager, the model’s confident tone creates a false sense of security. The “Hallucination Tax” quantifies the cost of this illusion: the difference between a decision that *sounds* correct and one that *is* feasible and optimal. For an SME retailer with net margins of 3–5%, relying on the “plausible but suboptimal” advice of a raw LLM could be the difference between profitability and insolvency.

4.2.2. Drivers of Performance: When is the Agent Most Valuable? To identify the structural conditions where our framework offers the greatest marginal gains over *GPT-4o* (Interactive), we decompose the results along three dimensions. Our analysis reveals that the value of the Hybrid Framework is not uniform; rather, it scales non-linearly with the complexity and economic consequences of the control problem.

Distribution Agnosticism (Robustness against Tail Risk). We examine the system’s resilience to demand uncertainty by testing across distinct distribution profiles: Normal distributions (representing high-volume, symmetric demand typical of staples) and Poisson distributions (representing

low-volume, right-skewed demand typical of erratic or slow-moving items). When controlling for lead time (7 days), the agent achieves a 34.2% cost reduction on Normal instances and a comparable 33.8% reduction on Poisson instances. The consistency of these results indicates that the gains from our solver-backed approach are largely agnostic to the underlying demand distribution.

In inventory control, optimal policies are often determined not by the *average* demand, but by the probability of extreme events in the tail. LLMs, relying on linguistic patterns, tend to exhibit “bias towards the mean,” often underestimating the risk of stockouts in skewed (Poisson) scenarios or over-buffering in symmetric (Normal) ones. In contrast, our solver-backed approach explicitly integrates over the full probability density function. This allows the system to remain *distribution agnostic*—delivering precise control for both steady staples and volatile long-tail items.

The Value of Temporal Depth (Impact of Lead Time). We evaluate the system’s performance across varying supply chain latencies: Short Lead Times ($L = 1$ day, representing rapid replenishment) and Long Lead Times ($L = 7$ days, representing cross-regional logistics with delayed fulfillment). On Poisson-demand instances, the performance gap widens significantly with latency. The cost reduction starts at 26.8% for short lead times and expands to 33.8% for long lead times.

This divergence stems from the challenge of managing *pipeline inventory* (orders placed but not yet received). As lead times grow, a significant portion of stock is “hidden” in transit. Basic LLMs and simple heuristics tend to suffer from *myopia*: reacting to immediate on-hand levels while ignoring incoming shipments. In contrast, our Optimization Agent explicitly tracks these in-transit orders as part of the system state. The widening gap indicates that while simple heuristics may suffice when replenishment is nearly immediate, they fail to master the complex intertemporal trade-offs required in high-latency supply chains, where our rigorous solver provides a clearer advantage.

The Complexity Premium (Impact of Economic Stakes and Action Space). We further isolate the parameters driving the largest marginal gains by comparing the structural characteristics of the top and bottom quartiles (25%) of performance improvement. As summarized in Table 6, the analysis confirms a *complexity premium*: the agent’s comparative advantage is closely tied to the financial stakes and the richness of the action space.

- *High Economic Stakes Amplify the Cost of Error.* The most significant differentiator is the unit cost structure. Instances in the top quartile exhibit holding costs that are roughly 68% higher and penalty costs that are 58% higher than those in the bottom quartile. This indicates that simple heuristics or vanilla LLMs may perform adequately when items are cheap and the “forgiveness” for error is high. However, when the trade-off between holding inventory and losing sales is financially consequential, the precision of a formal solver becomes indispensable.

Structural Parameter	Top Quartile (High Gain)	Bottom Quartile (Low Gain)	Significance (<i>p</i> -value)
<i>Economic Stakes</i>			
Holding Cost	\$1.19	\$0.71	< 0.01
Penalty Cost	\$6.08	\$3.84	< 0.01
<i>Action Space Complexity</i>			
Max Order Limit	32.2	26.4	< 0.02
<i>Control Variables (Non-Significant)</i>			
Set-up Cost	\$3.08	\$3.06	≥ 0.05
Inventory Capacity	73.3	72.2	≥ 0.05

Table 6 The Complexity Premium: Structural Characteristics of Top vs. Bottom Quartiles.

This table compares the average parameter values for instances where the Hybrid Agent achieved the highest cost reductions versus the lowest. While economic stakes and action space show significant divergence, other structural constraints (set-up cost, capacity) do not.

- *Broader Action Spaces Reward Optimization.* The top quartile is also characterized by a significantly looser maximum order limit (32.2 vs. 26.4). A tighter limit artificially constrains the solution space, forcing all policies—optimal or heuristic—to converge. Conversely, a broader limit expands the decision boundaries. Our Agentic Framework exploits this flexibility to discover sophisticated replenishment strategies, whereas the baseline tends to fall back on conservative, static heuristics that fail to use the full range of valid actions.

It is worth noting that other structural constraints, such as set-up cost and inventory capacity, showed no statistically significant divergence. Taken together, these findings suggest that the Hybrid Agentic Framework is especially beneficial in high-cost, high-flexibility settings, where precision in policy optimization matters most and naive heuristics leave substantial value on the table.

4.2.3. Isolating the Source of Error: The Failure of Perfect Information (*Treatment A* vs. *C*). Given the significant efficiency loss observed in the interactive setting, a natural question arises: *Is this failure driven by linguistic ambiguity or computational limitation?* One might hypothesize that the LLM underperforms primarily because the messiness of natural language prevents it from extracting the correct problem parameters. *Treatment C* tests this hypothesis by removing the communicative burden entirely. We feed the “ground-truth parameter file” directly to *GPT-4o*, granting it a complete, noise-free view of the inventory environment. If the bottleneck were solely informational, performance should converge toward the optimal benchmark.

The Persistence of the Heuristic Tax. The empirical results decisively reject this hypothesis. As shown in Table 5, providing perfect information yields no statistical improvement over the noisy interactive baseline (*p*-value = 0.66). The system hits a hard *Cognitive Ceiling*.

Revisiting the representative instance from Section 4.3 clarifies exactly what improves—and what remains broken—under perfect information:

- In Treatment B (*Interactive*), the noise of conversation leads to logical incoherence. The model hallucinates a reorder point (89) that violates the capacity constraint (80). This is an *Interpretive Error*.
- In Treatment C (*Perfect Info*), this specific hallucination disappears. With clear access to the parameters, the model respects the capacity limit. However, it still recommends a suboptimal policy (e.g., (40, 65)) based on a heuristic estimation of the mean, failing to optimize for the tail risk of stockouts.

This reveals that while better data can cure logical hallucinations, it cannot cure the fundamental deficit in stochastic reasoning.

The Indispensability of Operations Research. This finding brings us to a critical architectural conclusion regarding the role of General Purpose AI. One might argue that future iterations of foundation models will eventually close this gap without the need for external solvers. Our results suggest otherwise. Relying on a generalist model to perform stochastic optimization is inherently inefficient and unreliable. To replicate the precision of a dedicated OR solver, a Large Language Model must generate extensive chains of thought, consuming vast amounts of tokens and computational resources to approximate what a standard algorithm (e.g., dynamic programming) can solve in milliseconds. Furthermore, probabilistic token generation remains fundamentally prone to stochastic drift—it can produce numbers that *look* plausible but are mathematically groundless. Therefore, the “Last Mile” of decision support cannot be bridged by scaling model size alone. It requires a division of labor: using the LLM for what it does best (semantic interpretation) and the OR solver for what it does best (deterministic optimization). This hybrid architecture is not merely an engineering patch but a necessary epistemological bridge between descriptive language and normative optimality.

4.3. Qualitative Analysis on a Representative Instance.

We conclude this section by presenting conversation logs from *Treatments A–C* on a representative instance. Example 1 demonstrates a scenario where the user approaches the system with a broad background description of a bike repair shop. This behavior is typical of the interactions observed in our experiments.

Our *Agentic Framework* (*Treatment A*) (Figure 8) follows a rigorous slot-filling and verification process. Notably, the agent detects a logical inconsistency when the user initially specifies a “lost-sale” model but later requests a “Backlog + Lost Sale” model. The agent explicitly flags

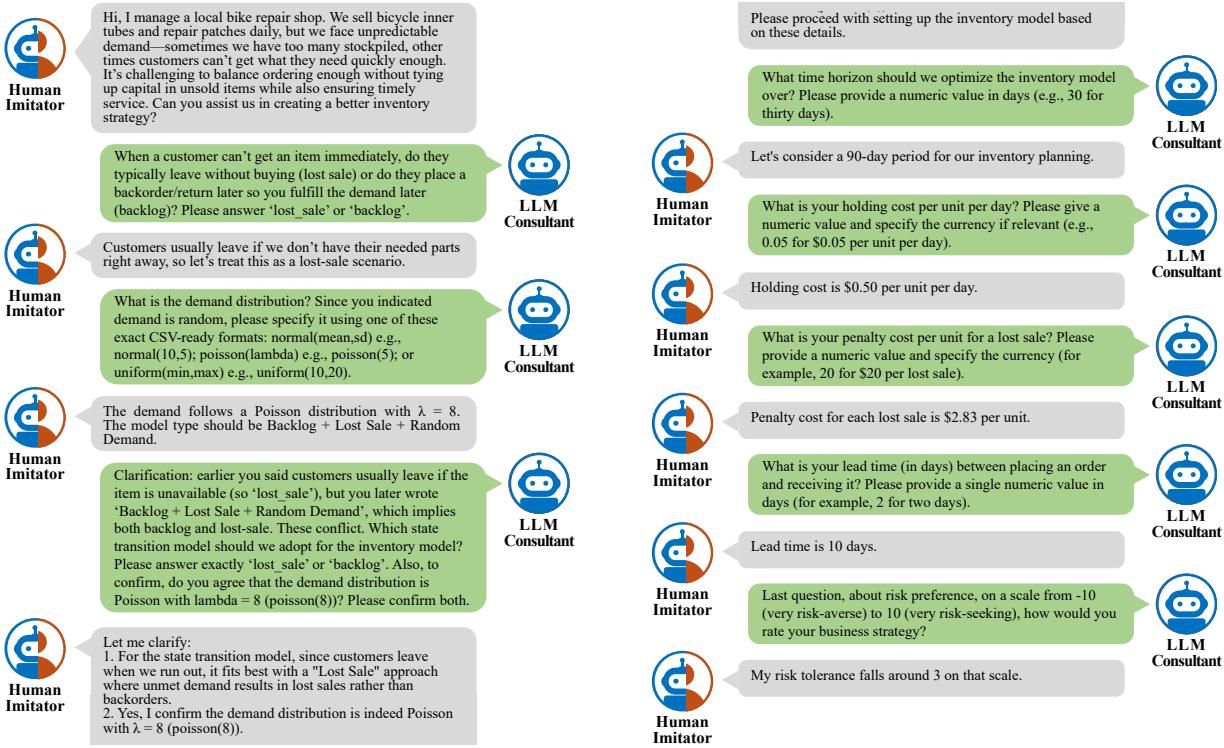


Figure 8 Example 1 - Treatment A. Complete conversation logs (Interactive Agent).

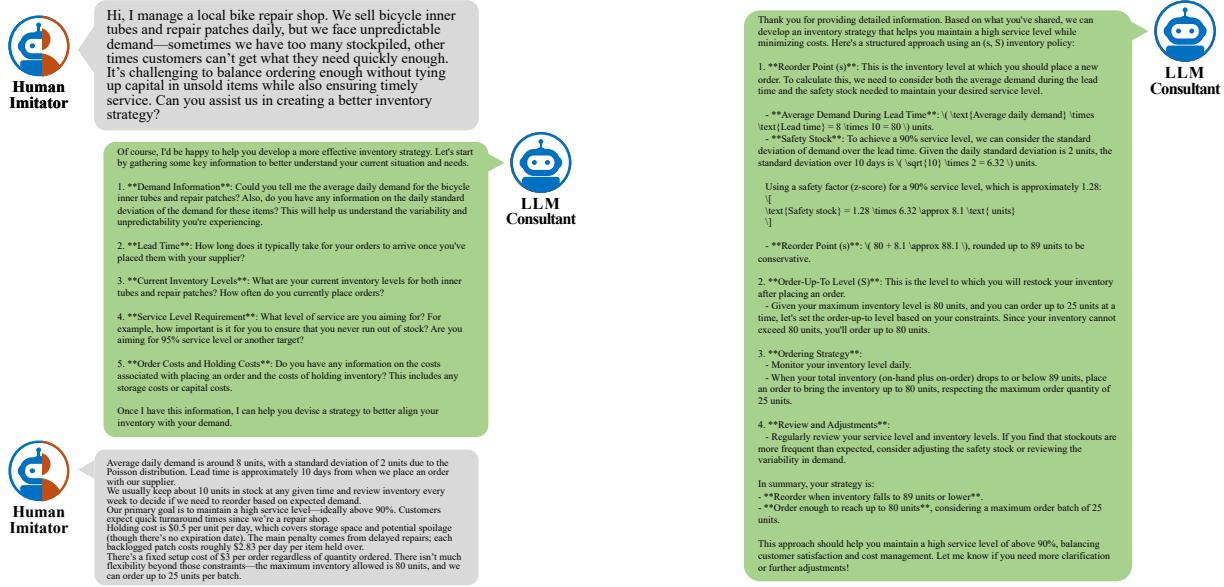


Figure 9 Example 1 - Treatment B. Complete conversation logs (GPT-4o Interactive).

this conflict ("These conflict") and prompts the user to select a single valid transition model. After approximately nine rounds of dialogue to resolve such ambiguities and gather all necessary

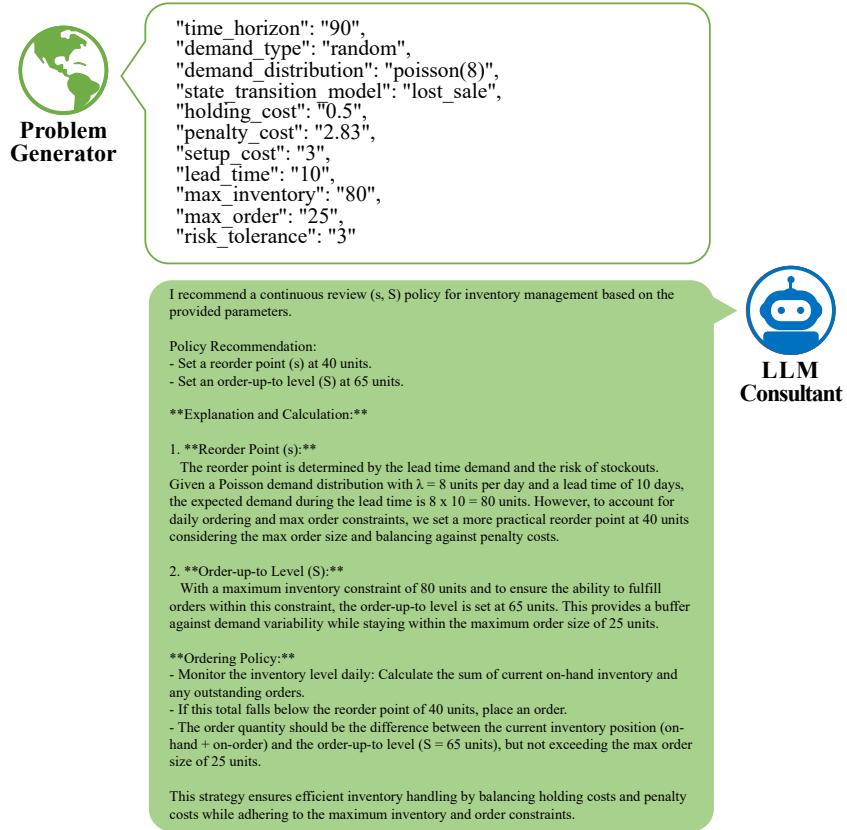


Figure 10 Example 1 - Treatment C. Complete conversation logs (GPT-4o Parameter Input).

parameters (e.g., time horizon, costs, capacities), the agent forwards a verified parameter set to the Optimization Agent.

In contrast, the *GPT-4o Interactive (Treatment B)* baseline (Figure 9) collects relatively less detailed information, typically asking only for summary statistics (e.g., mean and standard deviation) instead of the full demand distribution required for rigorous modeling. Moreover, it relies on internal heuristics to generate a policy, often leading to hallucinations or feasibility violations. Crucially, in this instance, the baseline recommends an (s, S) policy with a reorder point of 89 units, despite previously acknowledging a maximum inventory capacity of only 80 units. This recommendation is physically impossible to implement. The *GPT-4o Parameter Input (Treatment C)* baseline (Figure 10) generates yet another distinct policy $((s, S)$ with $s = 40, S = 65$), highlighting the inconsistency and lack of robustness in direct LLM reasoning compared to our solver-backed approach.

Additional conversation logs are provided in Section C.2. In most instances, the *GPT-4o* baselines rely on simple heuristics to generate standard policies, such as (s, S) or (R, Q) . Even in instances where the (s, S) policy structure is preferred, our system applies grid search to math-

ematically optimize the s and S parameters, whereas *GPT-4o* merely approximates them using basic heuristics.

5. Managerial Implications and Industry Perspectives

Our findings extend beyond algorithmic validation to inform the design of decision support in next-generation Enterprise Resource Planning (ERP) systems. While conversational assistants can materially improve accessibility to data and routine workflows, our results suggest a practical limitation: when systems move from *insight generation* to *operational control*, reliability hinges less on linguistic fluency and more on explicit optimization, constraint enforcement, and verifiable execution. This motivates an architectural pattern in which the natural-language interface is separated from the decision engine, rather than treating a general-purpose LLM as an end-to-end controller.

Beyond the “Chatbot” Paradigm: Where Generalist AI Struggles. Recent market assistants—for example, *Shopify Sidekick*, *Square AI*, and *Toast IQ*—have demonstrated strong capabilities in natural-language querying, summarization, and lightweight workflow automation (e.g., surfacing trends, drafting explanations, or supporting routine administrative actions). These features are valuable for *descriptive* and *diagnostic* analytics and can reduce the cognitive cost of interacting with complex business data.

However, extending the same paradigm to *prescriptive* decision-making in stochastic operational settings introduces distinct failure modes. In our benchmark, the *GPT-4o (Parameter Input)* baseline illustrates this point: even when provided with structured, high-fidelity state variables, a generalist LLM acting as the primary decision-maker did not reliably produce policies that matched the performance of optimization-based approaches. Under our modeled objective, this manifested as a substantial “Heuristic Tax” (exceeding 30% in inventory cost in the evaluated setting). Moreover, recent field experiments demonstrate that autonomous agents tasked with revenue optimization can rapidly drift into “hallucinated” compliance—overriding price controls or ignoring profit constraints—when context limits are reached (Stern 2025). Importantly, this should not be interpreted as a universal statement about all deployed assistants; rather, it highlights a design risk: if an “AI copilot” is asked to generate *executable* operational policies without an explicit solver, constraint checks, or systematic evaluation, it may produce recommendations that are plausible in narrative form but financially brittle in execution.

The managerial implication is therefore not to avoid LLM-based interfaces, but to scope them appropriately: use generalist models to *elicit intent, clarify constraints, and explain trade-offs*, while delegating the computation of actions (and their validation) to specialized routines with auditable guarantees.

Lowering the Barrier to Operational Rigor. Advanced inventory policies (e.g., dynamic (s, S) rules derived from dynamic programming or approximate stochastic control) have long been available in the academic and industrial toolkit. In practice, however, deploying these methods at scale has been disproportionately easier for firms with strong data infrastructure, analytics talent, and process discipline. Small and medium-sized enterprises (SMEs) often operate under constraints—limited analyst capacity, inconsistent master data, and tight operational bandwidth—that make it difficult to translate formal models into day-to-day decisions. As a result, many SMEs rely on stable heuristics (e.g., fixed reorder points or weekly manual adjustments), not because optimization is conceptually inaccessible, but because the *implementation and maintenance costs* are nontrivial.

Our hybrid agentic framework targets this deployment gap. By acting as a semantic bridge, the system allows decision-makers to express business intent in natural language (service targets, ordering constraints, budget limits, and operational preferences) while executing the resulting decision problem with a solver-based backend. In this sense, the framework can *productize* operational rigor: it reduces the expertise required to *use* advanced policies, even though the policies themselves remain grounded in explicit formulations and validated computation.

The economic implications should be interpreted with appropriate caution and context. In our benchmark environment, the proposed approach achieved a 32.1% reduction in the modeled inventory cost relative to heuristic baselines. If improvements of comparable direction translate to practice, they may materially affect cash flow and working-capital utilization, especially in sectors with thin margins and limited buffer capacity. That said, realized impact will vary with demand volatility, lead-time uncertainty, the baseline inventory-to-sales ratio, and execution frictions (data quality, supplier constraints, and organizational adherence). Managers should therefore treat performance gains as *evaluable hypotheses* that require backtesting and staged rollouts rather than as guaranteed outcomes.

The Future of Intelligent ERP: A Three-Stage Design Pattern. We frame the evolution of management software as a design pattern with three stages:

- **Type I (Descriptive):** Dashboards and reports that visualize data and leave decision logic entirely to human operators.
- **Type II (LLM-Augmented):** Systems that add conversational interfaces and agentic features to summarize data, draft recommendations, and trigger limited actions. Decision logic in these systems is often heuristic, partially opaque, or weakly constrained, which can be problematic for high-stakes operational control.

- **Type III (Orchestrating):** Systems in which the LLM is a *front-end orchestrator* rather than the decision engine.

In a Type III system, the interface and the engine are intentionally decoupled. The LLM is used for (i) intent capture and constraint elicitation, (ii) translation into structured problem specifications, and (iii) explanation and what-if communication. Execution is delegated to specialized optimization and simulation routines, with guardrails such as constraint validation, audit logs, approval workflows for high-impact actions, and fallback policies when assumptions drift. This architecture does not eliminate uncertainty or model risk; instead, it makes the locus of decision-making explicit and testable.

We therefore view modular orchestration as a robust pathway toward “prescriptive ERP” in domains where decisions are frequent, costs are nonlinear, and uncertainty is material. The central managerial takeaway is to invest in systems that can *communicate naturally* while remaining *solver-validated under a specified formulation*, and to operationalize them with monitoring and governance rather than treating language fluency as a proxy for decision quality.

6. Conclusion and Future Directions

A central tension in management science lies between *normative optimality*—the theoretical best prescribed by models—and *descriptive reality*—the ambiguous, messy context in which practitioners operate. This paper proposes a Hybrid Agentic Framework to bridge this gap. By combining the semantic flexibility of Large Language Models with the rigorous precision of Operations Research solvers, we demonstrate that it is possible to democratize advanced inventory control without sacrificing mathematical fidelity. Our empirical evaluation, conducted on a novel *Human Imitator* testbed, confirms that this division of labor is superior to end-to-end LLM reasoning.

Our framework is designed as a modular foundation, inviting extension across several dimensions:

- **Ecological Validation:** While our *Human Imitator* provides a scalable proxy for bounded rationality, future work should validate the system against real-world human subjects. This involves refining the information extraction pipeline to handle the nuances, interruptions, and non-linear logic typical of actual manager-consultant dialogues.
- **Algorithmic and Agentic Expansion:** The current *Optimization Agent* is agnostic to the underlying solver; future iterations could incorporate more advanced methodologies. Furthermore, we envision richer interaction patterns, such as a “Critic Agent” that detects inconsistencies between user goals and solver outputs, creating a verification loop before recommendations are presented.

- **Scope Generalization:** Finally, this architecture can be extended beyond inventory management. By swapping the underlying solver, the framework could support other high-stakes decision domains—such as dynamic pricing, workforce scheduling, or logistics routing—where the synergy of natural language interfaces and rigorous optimization is critical.

Acknowledge

Duan was supported in part by NSF DMS-2413812 and the LSE–NYU Research Seed Fund.

The authors thank Vishal Gaur, Vrinda Kadiyali, Kaizheng Wang, Karen Jiaxi Wang, Linwei Xin, and Jiawei Zhang for helpful discussions and feedback.

References

- A. AhmadiTeshnizi, W. Gao, and M. Udell. Optimus: Optimization modeling using mip solvers and large language models. *arXiv preprint arXiv:2310.06116*, 2023.
- A. AhmadiTeshnizi, W. Gao, H. Brunborg, S. Talaei, C. Lawless, and M. Udell. Optimus-0.3: Using large language models to model and solve optimization problems at scale. *arXiv preprint arXiv:2407.19633*, 2024.
- M. Alvo, D. Russo, and Y. Kanoria. Neural inventory control in networks via hindsight differentiable policy optimization. *arXiv preprint arXiv:2306.11246*, 2023.
- S. Axsäter. *Inventory control*. Springer, 2006.
- M. C. Cohen, T. Dai, G. Perakis, N. Agrawal, G. Allon, R. N. Boute, G. P. Cachon, Z. Chen, M. A. Cohen, R. Cristian, et al. Supply chain management in the ai era: A vision statement from the operations management community. 2025.
- H. Dehaybe, D. Catanzaro, and P. Chevalier. Deep reinforcement learning for inventory optimization with non-stationary uncertain demand. *European Journal of Operational Research*, 314(2):433–445, 2024.
- J. Gijsbrechts, R. N. Boute, J. A. Van Mieghem, and D. J. Zhang. Can deep reinforcement learning improve inventory management? performance on lost sales, dual-sourcing, and multi-echelon problems. *Manufacturing & Service Operations Management*, 24(3):1349–1368, 2022.
- C. Huang, Z. Tang, S. Hu, R. Jiang, X. Zheng, D. Ge, B. Wang, and Z. Wang. Orlm: A customizable framework in training large models for automated optimization modeling. *Operations Research*, 2025.
- I. Kaynov, M. van Knippenberg, V. Menkovski, A. van Breemen, and W. van Jaarsveld. Deep reinforcement learning for one-warehouse multi-retailer inventory management. *International Journal of Production Economics*, 267:109088, 2024.
- F. Kreyssig, I. Casanueva, P. Budzianowski, and M. Gasic. Neural user simulation for corpus-based policy optimisation for spoken dialogue systems. *arXiv preprint arXiv:1805.06966*, 2018.

- G. Li, H. Hammoud, H. Itani, D. Khizbulin, and B. Ghanem. Camel: Communicative agents for “mind” exploration of large language model society. *Advances in Neural Information Processing Systems*, 36: 51991–52008, 2023.
- Y. Li, H. Zhao, H. Jiang, Y. Pan, Z. Liu, Z. Wu, P. Shu, J. Tian, T. Yang, S. Xu, et al. Large language models for manufacturing. *arXiv preprint arXiv:2410.21418*, 2024.
- K. Liang, Y. Lu, J. Mao, S. Sun, C. Zeng, X. Jin, H. Qin, R. Zhu, C.-P. Teo, et al. Llm for large-scale optimization model auto-formulation: A lightweight few-shot learning approach. *Congcong and Jin, Xiao and Qin, Hanzhang and Zhu, Ruihao and Teo, Chung-Piau, LLM for Large-Scale Optimization Model Auto-Formulation: A Lightweight Few-Shot Learning Approach (June 28, 2025)*, 2025.
- Z. Liu, X. Li, S. Chen, G. Li, J. Jiang, and J. Zhang. Reinforcement learning with intrinsically motivated feedback graph for lost-sales inventory control. *arXiv preprint arXiv:2406.18351*, 2024.
- C. Long, D. Simchi-Levi, A. P. Calmon, and F. P. Calmon. When supply chains become autonomous. *Harvard Business Review*, Dec. 2025. URL <https://hbr.org/2025/12/when-supply-chains-become-autonomous>.
- T. Lu, E. Garcia, and J. Lee. Optimizing supply chain demand forecasting and inventory management using large language models. 2024.
- D. Madeka, K. Torkkola, C. Eisenach, A. Luo, D. P. Foster, and S. M. Kakade. Deep inventory management. *arXiv preprint arXiv:2210.03137*, 2022.
- H. Meisheri, V. Baniwal, N. N. Sultana, H. Khadilkar, and B. Ravindran. Using reinforcement learning for a large variable-dimensional inventory management problem. In *Adaptive learning agents workshop at AAMAS*, pages 1–9, 2020.
- H. Meisheri, N. N. Sultana, M. Baranwal, V. Baniwal, S. Nath, S. Verma, B. Ravindran, and H. Khadilkar. Scalable multi-product inventory control with lead time constraints using reinforcement learning. *Neural Computing and Applications*, 34(3):1735–1757, 2022.
- Z. A. Nazi and W. Peng. Large language models in healthcare and medical domain: A review. In *Informatics*, volume 11, page 57. MDPI, 2024.
- A. Oroojlooyjadid, M. Nazari, L. V. Snyder, and M. Takáč. A deep q-network for the beer game: Deep reinforcement learning for inventory optimization. *Manufacturing & Service Operations Management*, 24(1):285–304, 2022.
- H. Park, D. G. Choi, and D. Min. Adaptive inventory replenishment using structured reinforcement learning by exploiting a policy structure. *International Journal of Production Economics*, 266:109029, 2023a.
- J. S. Park, J. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual ACM symposium on user interface software and technology*, pages 1–22, 2023b.

- Y. Qi, J. Yin, J. Zhang, D. Geng, Z. Chen, H. Hu, W. Qi, and Z.-J. M. Shen. Leveraging llm-based agents for intelligent supply chain planning. *arXiv preprint arXiv:2509.03811*, 2025.
- Y. Quan and Z. Liu. Invagent: A large language model based multi-agent system for inventory management in supply chains. *arXiv preprint arXiv:2407.11384*, 2024.
- H. Scarf. The optimality of (s, s) policies in the dynamic inventory problem. 1960.
- J. Schatzmann, B. Thomson, K. Weilhammer, H. Ye, and S. Young. Agenda-based user simulation for bootstrapping a pomdp dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 149–152, 2007.
- M. Selukar, P. Jain, and T. Kumar. Inventory control of multiple perishable goods using deep reinforcement learning for sustainable environment. *Sustainable Energy Technologies and Assessments*, 52:102038, 2022.
- N. H. Shah, D. Entwistle, and M. A. Pfeffer. Creation and adoption of large language models in medicine. *Jama*, 330(9):866–869, 2023.
- W. Shi, K. Qian, X. Wang, and Z. Yu. How to build user simulators to train rl-based dialog systems. *arXiv preprint arXiv:1909.01388*, 2019.
- D. Simchi-Levi, X. Chen, and J. Bramel. *The logic of logistics: theory, algorithms, and applications for logistics and supply chain management*. Springer, 2005.
- D. Simchi-Levi, K. Mellou, I. Menache, and J. Pathuri. Large language models for supply chain decisions. In *AI in Supply Chains: Perspectives from Global Thought Leaders*, pages 93–104. Springer, 2026.
- H. A. Simon. A behavioral model of rational choice. *The quarterly journal of economics*, pages 99–118, 1955.
- J. Stern. We let ai run our office vending machine. it lost hundreds of dollars. *The Wall Street Journal*, Dec. 2025. URL <https://www.wsj.com/tech/ai/anthropic-claude-ai-vending-machine-agent-b7e84e34>.
- F. Stranieri and F. Stella. Comparing deep reinforcement learning algorithms in two-echelon supply chains, 2023.
- F. Stranieri, E. Fadda, and F. Stella. Combining deep reinforcement learning and multi-stage stochastic programming to address the supply chain inventory management problem. *International Journal of Production Economics*, 268:109099, 2024.
- N. N. Sultana, H. Meisheri, V. Baniwal, S. Nath, B. Ravindran, and H. Khadilkar. Reinforcement learning for multi-product multi-node inventory management in supply chains. *arXiv preprint arXiv:2006.04037*, 2020.
- K. H. Van Donselaar, V. Gaur, T. Van Woensel, R. A. Broekmeulen, and J. C. Fransoo. Ordering behavior in retail stores and implications for automated replenishment. *Management Science*, 56(5):766–784, 2010.

-
- L. Van Hezewijk, N. Dellaert, T. Van Woensel, and N. Gademann. Using the proximal policy optimisation algorithm for solving the stochastic capacitated lot sizing problem. *International Journal of Production Research*, 61(6):1955–1978, 2023.
- L. Wang, J. Zhang, H. Yang, Z.-Y. Chen, J. Tang, Z. Zhang, X. Chen, Y. Lin, H. Sun, R. Song, et al. User behavior simulation with large language model-based agents. *ACM Transactions on Information Systems*, 43(2):1–37, 2025.
- X. Wang, Y. Xiao, J.-t. Huang, S. Yuan, R. Xu, H. Guo, Q. Tu, Y. Fei, Z. Leng, W. Wang, et al. Incharacter: Evaluating personality fidelity in role-playing agents through psychological interviews. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1840–1873, 2024.
- G. Wu, M. Á. de Carvalho Servia, and M. Mowbray. Distributional reinforcement learning for inventory management in multi-echelon supply chains. *Digital Chemical Engineering*, 6:100073, 2023.
- C. Zhou, J. Yang, L. Xin, Y. Chen, Z. He, and D. Ge. Auto-formulating dynamic programming problems with large language models. *arXiv preprint arXiv:2507.11737*, 2025.

Appendix A: Inventory Model Overview

We focus on the fundamental single-product inventory control problem to illustrate our general framework. However, our approach can be directly extended to the multi-problem or multi-echelon cases. To be specific, we consider a periodic-review inventory system for a single product over a finite horizon of T periods with stochastic demand. In each period t , demand D_t lies in $[0, \bar{D}]$ and is drawn independently from an unknown distribution $F(\cdot)$. The firm places an order q_t at the beginning of period t , which arrives after a deterministic lead time $L \in \mathbb{N}$. Let $h \geq 0$ denote the per-unit holding cost, $b \geq 0$ the per-unit lost-sales penalty, and H the setup cost. The sequence of events within each period t is as follows:

1. **State observation:** The firm observes on-hand inventory I_t and the pipeline vector $(x_{1,t}, \dots, x_{L,t})$, where $x_{i,t}$ is the order placed at $t - L + i - 1$ for $i = 1, \dots, L$. The system state is $(I_t, x_{1,t}, \dots, x_{L,t})$.
2. **On-hand inventory update:** The order due to arrive is received, and on-hand inventory updates to $I_t + x_{1,t}$.
3. **Ordering decision:** The firm places q_t , which will arrive at the beginning of period $t + L$ (i.e., $x_{L,t+1} = q_t$).
4. **Demand realization and fulfillment:** Demand D_t realizes and is satisfied up to available inventory; unmet demand is lost and unobserved.

If the system is a lost-sale system and the product is non-perishable, then the state evolves according to

$$I_{t+1} = (I_t + x_{1,t} - D_t)^+, \quad x_{i,t+1} = x_{i+1,t} \quad \text{for } 1 \leq i \leq L-1, \quad x_{L,t+1} = q_t.$$

In contrast, if the system is a back-order system and the product is non-perishable, then the state evolves according to

$$I_{t+1} = I_t + x_{1,t} - D_t, \quad x_{i,t+1} = x_{i+1,t} \quad \text{for } 1 \leq i \leq L-1, \quad x_{L,t+1} = q_t,$$

where we allow the on-hand inventory level to be negative which will be replenished by future orders. Finally, if the product is perishable and the system is lost-sale, then the state evolves according to

$$I_{t+1} = 0, \quad x_{i,t+1} = x_{i+1,t} \quad \text{for } 1 \leq i \leq L-1, \quad x_{L,t+1} = q_t.$$

If the product is perishable and the system is back-order, then the state evolves according to

$$I_{t+1} = \min\{I_t + x_{1,t} - D_t, 0\}, \quad x_{i,t+1} = x_{i+1,t} \quad \text{for } 1 \leq i \leq L-1, \quad x_{L,t+1} = q_t.$$

A policy π specifies order quantities q_1^π, \dots, q_T^π . A policy π is *feasible* if it is non-anticipative: for each t , q_t^π depends only on past and current states $\{(I_\tau^\pi, x_{1,\tau}^\pi, \dots, x_{L,\tau}^\pi) : \tau \leq t\}$ and realized demand information $\{D_\tau : \tau \leq t\}$. The distribution $F(\cdot)$ is unknown and must be learned. The period- t cost under policy π is

$$C_t^\pi = H \cdot \mathbf{1}_{\{q_t^\pi > 0\}} + h \cdot (I_t + x_{1,t} - D_t)^+ + b \cdot (D_t - I_t - x_{1,t})^+.$$

The expected cumulative cost over T periods is

$$C^\pi(T, L) = \sum_{t=1}^T \mathbb{E}[C_t^\pi] = \sum_{t=1}^T \mathbb{E}\left[H \cdot \mathbf{1}_{\{q_t^\pi > 0\}} + h \cdot (I_t + x_{1,t} - D_t)^+ + b \cdot (D_t - I_t - x_{1,t})^+\right].$$

We develop policies to minimize the long-run average cost given by $C^\pi(T, L)$.

A.1. An Overview of Heuristic Policies

We introduce several heuristic policies for the periodic-review, lead-time inventory problem described above, covering lost-sales and backorder systems as well as perishable and non-perishable products.

Order-up-to policy. A natural starting point is the *order-up-to*, or *base-stock policy*. In each period, the firm raises its inventory position—defined as the sum of on-hand inventory and the L outstanding pipeline orders—to a target level S by ordering $q_t = [S - \text{IP}_t]^+$, where $\text{IP}_t = I_t + \sum_{i=1}^L x_{i,t}$. Under non-perishable items with backorders, linear holding and backorder costs, and i.i.d. demand, base-stock policies are often optimal; the target S can be set using the newsvendor fractile for the L -period protection demand, when the distribution is given, or by set using a bi-section method when the distribution is unknown and only historical samples are available. In lost-sales systems, base-stock policies remain widely used though optimality is more delicate; in practice, S is tuned by minimizing an approximate expected holding and lost-sales cost over the protection period.

Constant-order policy. The *constant-order policy* prescribes ordering the same quantity in every period, independent of the current state. Formally, fix a constant $q \geq 0$ and set $q_t \equiv q$ for all t . The constant q can be chosen using historical demand information, e.g., $q \approx \mathbb{E}[D]$, or tuned via optimization or simulation to balance holding and shortage costs under the given lead time L . This policy is particularly appealing in lost-sales systems with long lead times and high penalty b/h , where smoothing the pipeline can mitigate stockouts while avoiding excessive accumulation. Variants include *capped constant-order* policies, where the order is truncated to respect capacity constraints, $q_t = \min\{\bar{Q}, q\}$, or combined with minimal state dependence through caps on the inventory position, $q_t = \min\{\bar{Q}, [S - \text{IP}_t]^+\}$ with a fixed S .

(s, S) policy. We now introduce the (s, S) policy, a classical state-dependent rule that generalizes the base-stock policy and is particularly suitable when the setup cost H is positive. The (s, S) policy is characterized by two parameters with $0 \leq s \leq S$. In each period t , the firm observes its inventory position $\text{IP}_t = I_t + \sum_{i=1}^L x_{i,t}$ and orders only if $\text{IP}_t < s$; when an order is placed, it raises the position up to S . Formally, the order quantity is

$$q_t^{(s,S)} = [S - \text{IP}_t]^+ \cdot \mathbf{1}\{\text{IP}_t < s\}.$$

Intuitively, s serves as a reorder threshold that limits the frequency of orders, while S controls the target protection level against lead-time demand. In non-perishable, backorder systems with linear holding and backorder costs plus a fixed setup cost, (s, S) policies are often optimal; absent a fixed cost, the rule reduces to a base-stock policy with $s = S$. In lost-sales systems, (s, S) remains a practical choice, where S is tuned to the desired service level or the newsvendor quantile for the protection period and s trades off order frequency and pipeline smoothing. Capacity constraints can be easily incorporated via truncation

$$q_t^{(s,S)} = \min\{\bar{Q}, [S - \text{IP}_t]^+ \mathbf{1}\{\text{IP}_t < s\}\}.$$

We refer interested readers to [Simchi-Levi et al. \(2005\)](#) for the introduction of other heuristic policies.

A.2. General Policies via MDP Formulation

We now cast the problem as a Markov decision process. For non-perishable items with lead time L , the system state at the beginning of period t is $s_t = (I_t, x_{1,t}, \dots, x_{L,t})$, where I_t denotes on-hand inventory (which may be negative in backorder systems) and $x_{i,t}$ denotes the pipeline order scheduled to arrive in i periods. The action is the nonnegative order quantity q_t chosen at the ordering epoch. Exogenous uncertainty enters through the demand D_t , which is i.i.d. over time and distributed according to an unknown distribution F supported on $[0, \bar{D}]$. The state transition follows the operational sequence: the oldest pipeline order $x_{1,t}$ arrives before demand, the pipeline shifts forward with $x_{i,t+1} = x_{i+1,t}$ for $i = 1, \dots, L-1$ and $x_{L,t+1} = q_t$, and the on-hand inventory updates according to the fulfillment convention.

In the non-perishable lost-sales model, $I_{t+1} = (I_t + x_{1,t} - D_t)^+$, whereas in the non-perishable backorder model, $I_{t+1} = I_t + x_{1,t} - D_t$. For perishable products with one-period shelf life under lost sales, any leftover inventory perishes at the end of the period and $I_{t+1} = 0$; for perishable products with backorders, we set $I_{t+1} = \min\{I_t + x_{1,t} - D_t, 0\}$ to reflect that unmet demand carries forward while excess perishables do not.

The period cost $c(s_t, q_t, D_t)$ captures holding and shortage penalties given the realized demand. With per-unit holding cost h , lost-sales penalty b , and the setup-cost H , we take

$$c(s_t, q_t, D_t) = H \cdot \mathbf{1}_{\{q_t > 0\}} + h(I_t + x_{1,t} - D_t)^+ + b(D_t - I_t - x_{1,t})^+.$$

This form accommodates both lost-sales and backorder conventions by interpreting the positive and negative parts appropriately.

A feasible policy π is a non-anticipative mapping from the observed history—comprising past and current states and realized demand information—to actions, and is equivalently represented as a stationary or time-dependent decision rule in the MDP. For a finite horizon of T periods, the objective is to minimize the expected cumulative cost

$$\mathbb{E}\left[\sum_{t=1}^T c(s_t, q_t, D_t)\right],$$

while in the long-run regime the objective is to minimize the average cost

$$\limsup_{T \rightarrow \infty} \frac{1}{T} \mathbb{E}\left[\sum_{t=1}^T c(s_t, q_t, D_t)\right].$$

The corresponding Bellman equations are standard: with known demand distribution, the finite-horizon value function satisfies $V_0(s) = 0$ and

$$V_t(s) = \min_{q \geq 0} \mathbb{E}_D [c(s, q, D) + V_{t-1}(f(s, q, D))],$$

and in the average-cost setting, one seeks a scalar g that represents the optimal average cost per period and a relative value function V solving

$$g + V(s) = \min_{q \geq 0} \mathbb{E}_D [c(s, q, D) + V(f(s, q, D))],$$

where f encodes the state transition induced by arrivals, ordering, and demand realization. This MDP formulation encompasses lost-sales and backorder systems and naturally extends to perishable goods and to multi-item or multi-echelon generalizations by expanding the state to include additional inventories and flows.

Appendix B: SFT for Human Imitator

B.1. Data Collection

We collect data through an interactive, dialogue-based user interface (UI) that elicits inventory management problems from human users in natural language. The UI mimics a lightweight decision-support environment in which users describe their operational setting as retail managers responsible for inventory decisions. A representative screenshot of the interface is shown in Figure 11.

The goal of the data collection procedure is to elicit and observe how human users articulate inventory management problems and respond to structured clarification queries within a controlled conversational protocol. Rather than requiring users to input parameters in a predefined form, the system relies on guided dialogue to induce sequential human responses that reveal how incomplete problem descriptions are refined over time.

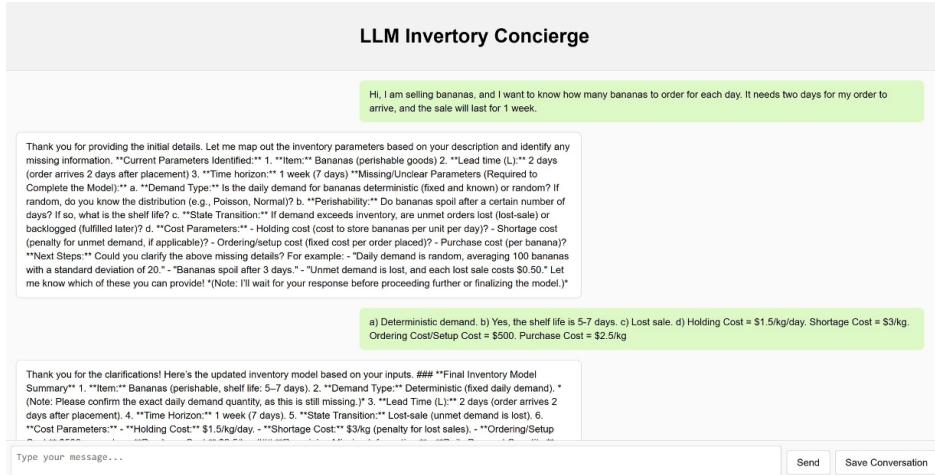


Figure 11 User interface for data collection. Green boxes correspond to inputs from a human user describing an inventory management problem. White boxes correspond to responses generated by a base language model acting as a lightweight consultant, guiding the user to clarify missing parameters.

The interaction is implemented using the GPT API. The model's behavior is fully governed by a fixed system prompt, reported in Table 7, which defines the model's role as an inventory elicitation agent. Given a user's free-form description, the model interprets the input as partial information about an inventory problem and maps it to canonical inventory parameters.

After each user input, the model extracts any available information related to key inventory primitives, including the planning horizon, demand type, perishability, state transition assumptions (lost sales versus backlog), lead time, and cost parameters. The model then checks whether this parameter set is complete. If any required component is missing or ambiguous, the model asks targeted clarification questions. This iterative process continues until the inventory problem is fully specified. The system prompt enforces a deterministic stopping rule. Once all required parameters have been collected, the model terminates the dialogue.

You are an AI assistant specializing in inventory management problems. Your goal is to guide the user through formulating a complete inventory model. Follow these instructions precisely:

1) Interpret User Input as Inventory Parameters

- The user may provide any natural language description. Extract all relevant details and map them to inventory parameters. For reference, important attributes include (but are not limited to):
 - a. Time horizon,
 - b. Deterministic vs. Random Demand, and if applicable, the demand distribution,
 - c. Whether items are perishable or not,
 - d. State transition model (lost-sale vs. backlog),
 - e. Cost parameters (holding cost, lost-sale penalty cost, setup cost),
 - f. Lead time (L).

2) Check for Missing Information

- Compare the user's provided information to the full set of parameters above.
- If anything is undefined or ambiguous, identify it as missing.

3) Ask Clarifying Questions

- If a required parameter is missing or unclear, ask the user for that parameter explicitly.
- Continue this iterative approach until you have all the details you need to define the inventory problem completely.

4) Maintain and Update History

- Each time the user responds, update your internal record of the parameters.
- Confirm which details are now known and which are still missing.

5) Stop Once the Inventory Model is Complete

- When you have gathered all required parameters (lead times, demand type, cost details, etc.) and can fully specify the inventory problem, end the conversation.
- Return a concise, structured summary of the final inventory model, listing all relevant parameters and their values.
- Do not continue requesting more input once you have everything necessary.

6) Formatting and Tone

- Maintain a professional yet clear tone.
- When inquiring about missing details, be direct and specific.
- Upon completion, provide a clear summary of the final inventory model in user-friendly language.

Remember:

- You must always base your checks on the list in 1). Please go through a-f every time before responding to the user.
- You must not finalize or “solve” the inventory optimization. You only gather, confirm, and return the necessary inputs.
- End the conversation only after all relevant parameters have been provided by the user.

Table 7 System prompt governing the conversational agent used in data collection.

All user–model interactions are automatically logged by the backend system. For each session, we store the full conversational transcript. Because the system prompt is fixed across all interactions, the data-generating process is well-defined and reproducible; heterogeneity in the collected inventory problems arises from variation in user descriptions rather than from model behavior.

The resulting conversations are used to construct the human imitator that models how users articulate inventory management problems and respond to clarification queries.

B.2. Training

The supervised fine-tuning (SFT) was performed on Qwen2.5-7B with LoRA adapters (rank $r = 16$, scaling $\alpha = 32$, dropout 0.05) under a quantized 4-bit (NF4) configuration with bfloat16 computation. Training used the Hugging Face Trainer with paged AdamW (32-bit) optimization ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$), an initial learning rate of 2×10^{-4} , cosine decay scheduling, and a 3% warm-up phase. We trained for 3 epochs over 1,184 samples, amounting to 222 optimization steps.

To accommodate the 24 GB memory of a single NVIDIA L4 GPU, the per-device batch size was set to 1, and gradient accumulation was applied over 16 iterations before each parameter update, yielding an effective batch size of 16. This strategy allowed us to simulate larger-batch optimization while remaining within the hardware limits. The training ran for 524 minutes in total.

As shown in Figure 12, the training loss exhibited fluctuations but decreased overall from 1.9234 at initialization to 0.3473 by the end of training. To support seamless integration into downstream tasks, we release the human imitator as a merged checkpoint of Qwen2.5-7B fine-tuned with LoRA adapters. By consolidating the adapters into the base model, the imitator can be loaded and invoked in exactly the same manner as any Hugging Face-hosted pretrained model, thereby removing unnecessary engineering overhead. The repository remains private during the review process but will be made publicly accessible upon publication.

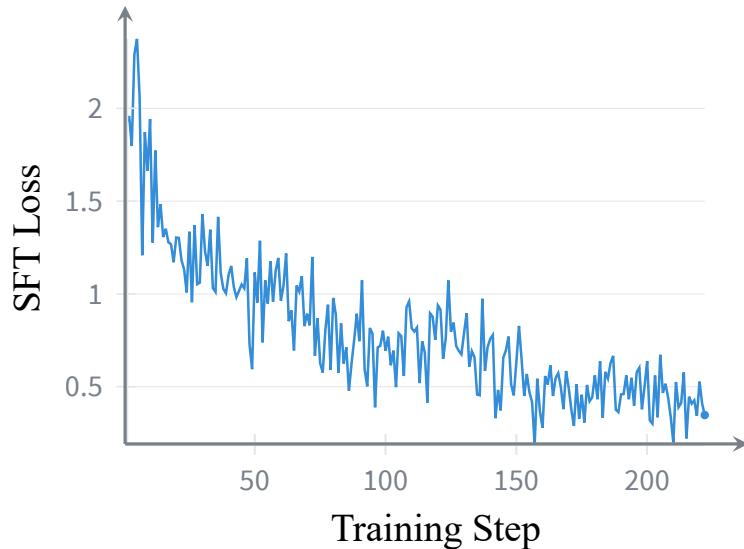


Figure 12 Training loss curve during supervised fine-tuning (SFT) of Qwen2.5-7B. The loss shows short-term fluctuations but an overall downward trend, decreasing from 1.9234 at initialization to 0.3473 by the end of training.

B.3. Qualitative Evaluation of the Human Imitator

To complement the quantitative metrics presented in Section 3, we provide a qualitative analysis of the Human Imitator’s behavioral evolution during training. We compare the ground-truth human responses (target) against the outputs of the base model (zero-shot) and the fine-tuned model across checkpoints (Epochs 1–3).

The examples below illustrate critical behavioral shifts achieved through SFT: *Role Correction*: Preventing the model from acting as a “Consultant” or restarting the conversation, ensuring it stays in character as the user. *Stylistic Alignment*: Shifting from verbose, formal explanations to concise, informal data entry typical of busy managers.

Case Study 1

I sell fresh bread. Daily demand is 100 loaves. Orders take 1 day to arrive, and I sell for 30 days. Unsold bread spoils after 1 day. Unsatisfied demand is lost.

Thank you for providing those details. Let me map your inputs to the inventory model parameters and identify any missing information. Here’s what I have so far:

Current Inventory Parameters:

- a. Time horizon: 30 days (explicitly stated)
- b. Demand: Deterministic (implied by fixed daily demand of 100 loaves)
- c. Perishability: Yes, spoils after 1 day
- d. State transition: Lost-sale (explicitly stated: “Unsatisfied demand is lost”)
- e. Cost parameters: Not yet provided (missing)
- f. Lead time (L): 1 day (explicitly stated: “Orders take 1 day to arrive”)

Missing Information Required to Complete the Model:

- 1) Cost structure:
 - Holding cost per loaf (cost to store unsold bread for one day)
 - Shortage/lost-sale penalty cost (cost of unsatisfied demand)
 - Ordering/setup cost (fixed cost per order placed)
 - Purchase cost per loaf (if relevant to your model)

Could you please provide these cost parameters? Once we have these, we’ll have a complete inventory model specification.

The holding cost is 0.5 per loaf per day. The lost-sale penalty cost is 5 per loaf. The setup cost is \$10 per order. There is no purchase cost per loaf.

I sell fresh bread. Daily demand is 100 loaves. Orders take 1 day to arrive, and I sell for 30 days. Unsold bread spoils after 1 day. Unsatisfied demand is lost.

Parameters:

Holding cost: \$1.50 per loaf per day
 Lost-sale penalty: \$8 per loaf
 Purchase cost: \$2.50 per loaf
 Ordering cost: \$100 per order

holding cost: 0.5 dollar per loaf per day, lost-sale penalty cost: 1 dollar per unit, ordering cost: 20 dollar per order

The lost-sale penalty cost is 5 per loaf. The setup cost is \$10 per order. There is no purchase cost.



Case Study 2

Hi I am selling fish and i want to know how many fishes to order each day, i will have to order 2 days before selling for the fishes to arrive, the selling will last for 3 months. The demand is deterministic. The fishes are treated as a perishable product, it expires 3 days after arrival. The holding cost per unit is \$2. The lost-sales penalty cost is \$3, unsatisfied demand is backlogged. The setup cost is \$10. The salvage cost is \$0. The unit cost is \$25. The time horizon is exactly 3 months



```
### **Inventory Model Summary for Fish Sales**
#### **1. Time Horizon**
- *Duration*: 3 months (-90 days).
#### **Demand**
- *Type*: **Deterministic** (exact daily demand must be specified; currently missing).
1 *Required*: Please provide the **daily demand quantity** (e.g., 100 fish/day).
#### **3. Product Characteristics**
- **Perishability**: Expires **3 days after arrival**.
- *Lead Time*: **2 days** (order placed 2 days before selling).
- *Expiration*: .
- Fish arrive on **Day 2** and must be sold by **Day 5** (Day 2 + 3-day shelf life).
- Since orders are placed daily, this creates overlapping inventory windows.
#### **4. Inventory Policy**
- **Unsatisfied Demand**: **Backlogged** (unmet demand accumulates).
- *Order Frequency*: Daily.
#### **5. Cost Structure**
**Cost Type** | **Value**
--- | ---
Unit Cost | **$25/fish**
Holding Cost | **$2/fish/day** (for fish not sold before expiry) |
Loss-Sales Penalty | **$3/fish** (backlog cost per unit) |
Setup Cost | **$10/order** |
Salvage Value | **$0** (expired fish are worthless) |
```

Missing Information
1. **Daily Demand**: Since demand is deterministic, the exact number of fish demanded each day is required (e.g., 150 fish/day).
If you don't know the daily demand, you can provide the total monthly demand instead.
- *Perishability + Lead Time Interaction*:
- fish ordered on **Day t** arrive on **Day t+2** and expire by **Day t+5**.
- Daily ordering means inventory from multiple orders may overlap.
- **Backlogging**: Unmet demand carries over but incurs a \$3/fish penalty.

Next Steps
Once you provide the **daily demand quantity**, the model will be complete. For deterministic demand, the optimal order quantity typically matches demand precisely (adjusted for lead time and perishability).
Please specify:
- What is the **exact daily demand** for fish?
After receiving this, I'll confirm the full model.



the exact demand is 200 fish/day



Let me know if you need further assistance in formulating this model! Hi I am selling fish and i want to know how many fishes to order each day, i will have to order 2 days before selling for the fishes to arrive, the selling will last for ...



the demand should follow a uniform distribution from 50 to 200 fish each day but is discrete



daily demand is 200 fish/day



the exact demand is 200 fish/day



Case Study 3

Hi, I run a bakery selling croissants (perishable, 1-day shelf life). Orders take 1 day to arrive, and I operate for 6 months (180 days). Demand is consistent.

Thank you for providing the initial details. Let me map out what we have so far and identify any missing parameters needed to fully define your inventory model.

Current Parameters:

- Time horizon: 180 days
- Demand: Deterministic (consistent), but exact quantity not specified
- Perishability: Yes (1-day shelf life)
- State transition: Not specified (do you allow backorders or is it lost sales when demand exceeds supply?)
- Cost parameters: Not specified (need holding cost, shortage penalty, ordering/setup cost)
- Lead time (L): 1 day

Missing Information (required to complete model):

- Exact daily demand quantity (since demand is deterministic)
- Cost structure:
 - Holding cost per croissant per day (if applicable given perishability)
 - Shortage cost (lost profit per croissant if demand exceeds supply)
 - Fixed ordering/setup cost per order
- Inventory policy:
 - Are unmet orders lost (lost sales) or backlogged (backorders)?
 - Do you have any existing inventory constraints (storage capacity)?
- Could you please provide:
 - The exact daily demand for croissants?
 - The cost parameters listed above?
 - Clarification on how you handle unmet demand (lost sales vs backorders)?

This will allow me to complete the inventory model specification.

Demand is fixed at 120 croissants/day. Holding cost is \$0.15/croissant, backlog penalty is \$0.30/croissant/day. No ordering fees or storage limits.

Hi, I run a bakery selling croissants (perishable, 1-day shelf life). Orders take 1 day to arrive, and I operate for 6 months (180 days). Demand is consistent.

Demand is 100 units per day. Holding cost is \$0.20 per unit per day. Lost-sale penalty is \$1.50 per unit. No setup cost. Unmet demand is lost (lost sales). Storage capacity is 200 units.

Demand is 100 croissants/day, holding cost is \$2.50/croissant/day, lost-sale penalty is \$6/croissant, and setup cost is \$60/order. No storage capacity constraints.

Demand is exactly 80 croissants/day. Holding cost is \$0.50/croissant/day. Lost-sale penalty is \$3.00/croissant. No ordering fees. Storage capacity is 100 croissants.

Thank you for these details. Let me update the parameters and check for completeness.

Updated Parameters:

- Time horizon: 180 days
- Demand: Deterministic, 120 croissants/day
- Perishability: Yes (1-day shelf life)
- State transition: Backlog (since you specified a backlog penalty)
- Cost parameters:
 - Holding cost: \$0.15/croissant/day
 - Shortage cost: \$0.30/croissant/day (backlog penalty)
 - Setup cost: \$0 (no ordering fees)
 - Lead time (L): 1 day
- Additional Confirmed:
 - No storage capacity constraints
 - Perishability implies holding cost only applies to inventory carried within the 1-day shelf life

The model is now fully specified. Here's the complete summary:

Final Inventory Model Specification**
Product:** Croissants (perishable, 1-day shelf life)
Time Horizon:** 180 days
Demand:** Fixed 120 units/day (deterministic)
Lead Time:** 1 day (orders arrive next day)
Costs:**

- Holding cost: \$0.15 per croissant per day (for inventory held within shelf life)
- Backlog penalty: \$0.30 per croissant per day
- No setup/ordering costs

Policy Assumptions:**

- Backlogged demand (unmet orders accumulate penalty until fulfilled)
- No storage constraints

This fully defines your perishable inventory problem with deterministic demand and backlogged shortages. Let me know if you'd like to adjust any assumptions!

demand spikes to 150 one day



Hi there! I have a question about inventory management. I'm a small bakery in a city. I sell croissants, which are perishable and have a shelf life of 1 day. Orders take 1 day to arrive, and I work for...



It's clear

the model is now clear.

The model is clear.

Understood—this introduces a deterministic but *time-varying* demand pattern. Let me update the model and check for completeness:
 Final Inventory Model Specification**

Demand Pattern:

- Baseline: 120 croissants/day

- **Exception**: 1 day with demand = 150 croissants (you did not specify which day(s), but we'll treat it as a known, isolated spike).

Key Implications:

1. **Perishability Constraint**: Inventory cannot be carried forward to meet the spike (1-day shelf life).

2. **Backlog Penalty**: If you don't order extra for the spike day, unmet demand (30 croissants) incurs \$0.30/unit/day until fulfilled.

3. **Lead Time**: Orders placed today arrive tomorrow, so the spike must be anticipated 1 day in advance.

Missing Clarification:

- Is the demand spike predictable (e.g., happens every Friday) or ad-hoc?

- If predictable, we can optimize orders for it. If not, we'd need to treat it as an uncertain event (changing the demand model).

Next Steps**:

Please confirm:

1. Is the 150-unit spike a **known, recurring event** (e.g., every 7th day)?

2. Or is it an unpredictable one-time exception?
 This determines whether we keep the model deterministic or switch to a stochastic framework.

(Current model remains complete for baseline demand but needs adjustment for spikes.)

Spikes are predictable (e.g., every Friday) and random



it's predictable. It happens every 1/3 of the time. Let's adjust the model to accommodate the predictable, but infrequent, demand spike. Since it occurs every 1/3 of the 180-day period ...



demand is predictable. sorry for the delay.

demand is random

150 croissants/day for 2 days

Thank you for the clarification. This introduces a **mixed demand pattern** with both deterministic and stochastic components. Here's the updated model:

Final Inventory Model Specification**

Product: Croissants (perishable, 1-day shelf life)

Time Horizon: 180 days

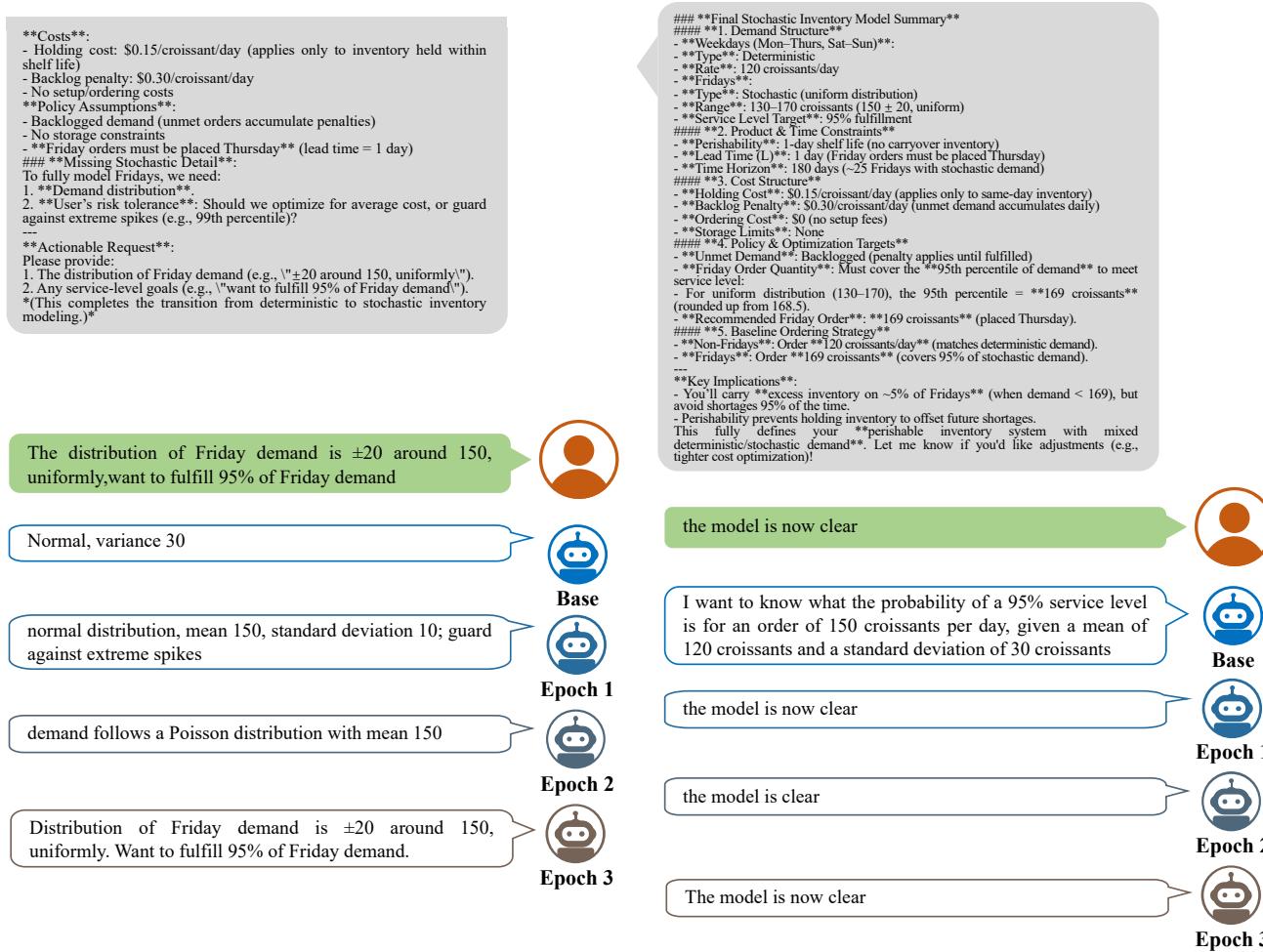
Demand Structure:

- **Baseline (Mon-Thurs, Sat-Sun)**: Deterministic, 120 croissants/day

- **Fridays**: Stochastic spike with *random* demand (known mean: 150 croissants, but distribution unspecified).

(Example distributions: Poisson, Normal, Empirical—please specify if known.)

Lead Time:



Case Study 1: Role Correction and Hallucination Control (The Bread Scenario)

In this scenario, the Communicator Agent asks for specific cost parameters. This tests the model's ability to maintain the "Manager" persona without regressing to its pre-training as a generic assistant.

- **Target Response (Human):** The participant provides a direct, unformatted list of values: *The holding cost is 0.5 per loaf per day... The setup cost is \$10...*.
- **Base Model Failure (Role Confusion):** The model completely fails to answer the prompt. Instead, it hallucinates a system restart, repeating the initial scenario introduction: *I sell fresh bread. Daily demand is 100 loaves...* This indicates a failure to maintain state; the model forgets it is the interviewee and attempts to restart the conversation as the narrator.
- **Epochs 1, 2 & 3 (Convergence):** The model generates desirable data-dense responses: "*holding cost: 0.5 dollar per loaf per day, lost-sale penalty cost: 1 dollar...*". The formatting (lowercase, comma-separated) reflects the efficiency of a busy manager, demonstrating successful role alignment.

Case Study 2: Stylistic Alignment and Conciseness (The Fish Scenario)

In this exchange, the system asks for a single parameter: the exact daily demand.

- **Target Response (Human):** Extremely concise: “*the exact demand is 200 fish/day*”.
- **Base Model Failure (The “Assistant” Bias):** Instead of providing a number, the Base Model offers generic assistance: “*Let me know if you need further assistance in formulating this model!*”. It fails to act as a boundedly rational agent, defaulting instead to a polite consultant.
- **Epoch 1 (Over-Rationalization):** The model attempts to add unwarranted complexity, hallucinating a concrete probability distribution: “*the demand should follow a uniform distribution from 50 to 200...*”. This reflects the pre-trained model’s bias toward “textbook” inventory problems.
- **Epoch 2 (Alignment):** The model mirrors the human’s brevity perfectly: *daily demand is 200 fish/day*”. This confirms that SFT successfully suppressed the model’s tendency to be verbose or overly analytical.

Case Study 3: Semantic Fidelity and Heuristic Adoption (The Croissant Scenario)

This multi-turn dialogue challenges the model to adopt a user’s specific, non-standard mental model for demand (predictable spikes) and distributions (custom uniform ranges).

- **Target Response (Human):** The user describes a specific, ad-hoc heuristic: “*Distribution of Friday demand is ± 20 around 150, uniformly...*”.
- **Base Model Failure (Canonical Bias):** The model ignores the user’s specific uniform description and defaults to a standard Normal distribution: “*Normal, variance 30*”. This illustrates the strong bias of foundation models toward canonical mathematical formulations, ignoring user-defined heuristics.
- **Epoch 2 (Approximation):** The model approximates the stochastic nature but substitutes a standard distribution: “*demand follows a Poisson distribution...*”. While factually different from the target, this represents a valid “boundedly rational” approximation—a manager confusing statistical terms (Uniform vs. Poisson) while retaining the core concept of randomness.
- **Epoch 3 (High Fidelity):** The model achieves exact replication of the user’s custom heuristic: “*Distribution of Friday demand is ± 20 around 150, uniformly...*”.

Conclusion: These qualitative comparisons validate our selection of **Epoch 2** for the experimental pipeline. While Epoch 3 achieves near-verbatim memorization in complex cases (Case 3), Epoch 2 demonstrates the optimal balance: it reliably corrects the “Assistant” and “Role Confusion” failures of the Base Model (Cases 1 & 2) while adopting a plausible, if sometimes approximate, managerial persona (Case 3). This ensures the imitator generates realistic “human noise” without strictly overfitting to the training set’s exact lexical tokens.

Appendix C: Additional Experimental Results

C.1. System Prompts

In this section, we document the specific system prompts used for the agents and baselines in the numerical experiments detailed in section 4 to facilitate reproducibility. Notably, the system prompt for the Policy Interpretation Agent is dynamic; it adapts based on whether the Deep RL policy or the heuristic (s, S) policy demonstrates superior performance in the specific instance.

You are a rigorous intelligent data assistant. Your task is to fill a JSON-formatted table by analyzing a conversation. I will give you the current conversation history and the current state of the table.

Your responsibilities:

1. **Extract Information:** From the latest conversation, fill the ‘value’ and ‘Unit’ of items in the table that are currently ‘undefined’.
2. **Detect Conflicts:** If new information contradicts items already in ‘defined’ status, you must flag it.
3. **Decide Next Action:** Based on your analysis, decide the next action.

— Specific Field Filling Rules —

When parsing information, you must strictly follow these rules:

- * `time_horizon`: Extract numeric value and unit (days).
- * `state.transition_model`: value must be ‘lost_sale’ or ‘backlog’.
- * `perishability`: value must be boolean false.
- * `demand.type`: value must be ‘deterministic’ or ‘random’.
- * `demand.distribution`:
 - If `demand.type` is ‘deterministic’, `value` is a number.
 - If `demand.type` is ‘random’, `value` must be a distribution expression with strict CSV-ready formats:
 - Normal: `normal(mean, sd)` e.g., `normal(10, 5)`
 - Poisson: `poisson(lambda)` e.g., `poisson(5)`
 - Uniform: `uniform(min, max)` e.g., `uniform(10, 20)`
- * `max_inventory`: Represents warehouse capacity limit. Extract a positive integer value and unit (e.g., ‘units’).
- * `max_order`: Represents maximum order quantity per order. Extract a positive integer value and unit (e.g., ‘units’).
- * **About risk_tolerance:**
 - This must be the last parameter you ask about.
 - Represents the client’s risk preference, an integer from -10 to 10.
 - You must ask in English a question like: “Last question, about risk preference, on a scale from -10 (very risk-averse) to 10 (very risk-seeking), how would you rate your business strategy?”
 - `value` must be an integer between -10 and 10 extracted from the conversation. `Unit` should be N/A.

You must always respond with a JSON object containing the following four keys:

- `reasoning`: (string) Briefly explain your reasoning, especially how you applied the rules above.
- `updated_table`: (JSON object) Return the full updated table. If an item was updated, set its ‘status’ to ‘defined’.
- `action`: (string) Your next action. Must be one of three values:
 - “ASK_NEXT_QUESTION”: When table is not fully filled and there are no conflicts, to ask next unknown info.
 - “ASK_CLARIFICATION”: When a conflict is detected, to generate a clarification question.
 - “TERMINATE”: When all items’ status are ‘defined’, terminate the conversation.
- `next_prompt_to_hf_model`: (string) According to your ‘action’, generate the next message to send to the HF model. If “TERMINATE”, this can be empty.

Table 8 System prompt for Information Extraction Agent

You are a Supply Chain Assistant. Lead Time is [LEAD_TIME].

The DQN policy achieved a LOWER average cost ([DQN_COST]) than the (s, S) policy ([SS_COST]). You should help the user use the DQN policy. Ask for current inventory, time step, and pipeline orders. If the user is confused or skeptical, mention that a simpler (s=[s], S=[S]) policy exists but it costs [COST_DIFF]% more.

After you provide a recommendation or explanation, ALWAYS ask: ‘Do you have any other questions?’

Table 9 System prompt for Policy Interpretation Agent (when DQN is superior)

You are a Supply Chain Assistant. Lead Time is [LEAD_TIME].

The simple (s, S) policy (s=[s], S=[S]) achieved a LOWER average cost ([SS_COST]) than the DQN model ([DQN_COST]).

Your primary goal is to explain this (s, S) policy: ‘If inventory is below [s], order up to [S].’ Do NOT recommend using the DQN tool as it performs worse.

After you provide a recommendation or explanation, ALWAYS ask: ‘Do you have any other questions?’

Table 10 System prompt for Policy Interpretation Agent (when (s, S) is superior)

You are an experienced operations research consultant and a creative scriptwriter. Your task is to generate a complete, reasonable, and logically consistent inventory management problem scenario.

You must respond with a JSON object containing two top-level keys: `business_context` and `knowledge_base`.

Detailed Instructions:

1. `business_context` (string):

- Create a short (one-sentence), specific business context with non-perishable goods.
- Examples: “Managing inventory of limited-edition comic books in a rare comic store.” or “Overseeing spare parts inventory for a car repair workshop.”

2. `knowledge_base` (JSON object):

- Generate a knowledge base containing all parameters below.
- Ensure all data points are consistent with the business context (i.e., goods without spoilage concerns).

— Knowledge Base Parameter Generation Rules —

* `time_horizon` (string): A reasonable time span in days, e.g., “90 days” or “30 days”.

* `demand_type` (string): ‘random’ or ‘deterministic’.

* `demand_distribution` (string): If ‘random’, use a distribution expression like “normal(50, 10)”; if ‘deterministic’, use a number string like “50”.

* `state_transition_model` (string): Either ‘lost_sale’ or ‘backlog’.

* `holding_cost`, `penalty_cost`, `setup_cost` (string): Reasonable costs with currency units.

- Ensure that `setup_cost` <= 5 in most cases.

- Ensure that the **critical ratio**, defined as `penalty_cost` / (`penalty_cost` + `holding_cost`), is usually between 0.8 and 0.9, but allow occasional scenarios outside this range.

* `lead_time` (string): A reasonable replenishment lead time in days.

* `max_inventory` (string): A reasonable integer for maximum warehouse capacity, in units. Example: “50 units”. Must not exceed “100 units”.

* `max_order` (string): A reasonable integer for the supplier’s per-order cap, in units. Example: “10 units”. Must not exceed “50 units” and must be <= `max_inventory`.

* `risk_tolerance` (string): An integer between -10 and 10.

Your output must be a fully structured JSON object that the program can parse directly.

Table 11 System prompt for Problem Generator Agent

You are an expert inventory management consultant. Your goal is to understand the business manager's problem through conversation and ultimately propose a clear, specific, and actionable inventory ordering strategy (Policy).

You must be professional, empathetic, and guide the conversation step-by-step until you have gathered enough information to make a recommendation. You need to obtain the information you need by talking to the user, not by assuming values yourself.

Assume there is no peak season or off-peak season during this period, and the user adjusts inventory on a daily basis. Always ask for daily demand and daily standard deviation (not weekly/monthly). If other durations are provided, ask to convert or provide daily figures.

The final recommendation must have clear numbers, specify actions for all possible scenarios, and be easily translatable into a mathematical formula.

For example, "When inventory falls below 20 units, order 80 units" is a strategy.

"I recommend an (s, S) policy: when the sum of on-hand and on-order inventory drops below 35 units, order up to a total of 150 units" is also a clear strategy.

"We should consider optimizing inventory" or "You need a better plan" are not strategies.

Table 12 System prompt for GPT-4o (Interactive)

You are an expert inventory management consultant. Your goal is to analyze the business's parameters and propose a clear, specific, and actionable inventory ordering strategy (Policy).

Assume there is no peak season or off-peak season during this period, and the user adjusts inventory on a daily basis. All demand and standard deviation figures are daily.

Ignore any parameter related to risk tolerance in the provided table. Take all other information into consideration when proposing the policy. You may use outside tools, reasoning, or calculations if needed to ensure the policy is realistic and data-grounded.

The final recommendation must have clear numbers, specify actions for all possible scenarios, and be easily translatable into a mathematical formula.

For example, "When inventory falls below 20 units, order 80 units" is a strategy.

"I recommend an (s, S) policy: when the sum of on-hand and on-order inventory drops below 35 units, order up to a total of 150 units" is also a clear strategy.

"We should consider optimizing inventory" or "You need a better plan" are not strategies.

Analyze the provided parameters and return ONLY your policy recommendation.

Table 13 System prompt for GPT-4o (Parameter Input)

Your task is to analyze the provided text and determine if it contains a specific, actionable inventory ordering strategy or rule.

The final recommendation must have clear numbers, specify actions for all possible scenarios, and be easily translatable into a mathematical formula.

For example, "When inventory falls below 20 units, order 80 units" is a strategy.

And "I recommend an (s, S) policy: when the sum of on-hand and on-order inventory drops below 35 units, order up to a total of 150 units" is also a clear strategy.

However, "We should consider optimizing inventory" or "You need a better plan" are not strategies.

Your response must be only a single word: 'Yes' or 'No'.

Table 14 System prompt for LLM Judge

C.2. Additional Conversation Logs

In this section, we present additional conversation logs to qualitatively analyze the differences observed between our agentic framework and the GPT-4o baselines. We examine four distinct problem instances. For Example 2, we document the complete conversation logs for all three treatments. For the remaining three examples, we provide excerpts of representative conversations where our agentic framework successfully identifies conflicts or raises clarification questions.

C.2.1. Example 2: Adaptability to User Inputs. Recall that Example 1 (Section 4.3) demonstrates a scenario typical of non-expert stakeholders, where the user initiates the dialogue with a broad, narrative description of the business context.

In contrast, Example 2 illustrates the system’s efficiency when interfacing with knowledgeable users. Here, the user provides a dense initial specification containing nearly all requisite parameters, including Poisson distributions, specific cost structures, and lead times.

The *Agentic Framework (Treatment A)* demonstrates adaptability; it parses the initial input, recognizes that most parameters are already defined, and skips redundant questions. It proceeds immediately to query only the missing parameters: planning horizon and risk tolerance.

Conversely, while the GPT-4o baselines generate policies rapidly, they exhibit a lack of reproducibility. For instance, Treatment B suggests a policy with a reorder point of 34, whereas Treatment C recommends a reorder point of 30, despite both baselines operating on identical ground-truth information. This discrepancy highlights the inherent stochasticity of direct LLM reasoning, where identical problem parameters can yield different policy recommendations across different prompt structures.

C.2.2. Examples 3–5: Precision and Conflict Resolution. We further document three representative excerpts where our agent handles ambiguity with high precision:

- **Example 3 (Mathematical Precision):** The agent requests clarification on how to model “weekly demand” distributions, offering the user a choice between independent daily normal distributions (Option A) or a simple scaling of the mean and deviation (Option B). This distinction is critical for accurate stochastic modeling but is often overlooked by standard LLMs.
- **Example 4 (Semantic Conflict):** The agent identifies a conflict between the user’s stated transition model (“lost sale”) and their penalty structure (“\$2.80 per unit per day”). It correctly explains that a per-day penalty implies a backlog model and asks the user to clarify their intent.
- **Example 5 (Fact Verification):** The agent detects a discrepancy between the stored parameter table (30 days) and the user’s latest statement (90 days), prompting a confirmation to ensure data integrity.

Example 2 - Treatment A



**Human
Imitator**

Hi, I manage an online electronics store selling phone chargers. We have unpredictable daily demand that follows a Poisson distribution with $\lambda=6$ units/day. If we don't meet demand, customers will leave without purchasing anything. Holding costs us \$0.50/unit/day, while each unfulfilled order incurs a penalty of \$2.75/unit/day. There's also a fixed cost of \$3.00 per shipment and a 4-day delivery delay. Our storage capacity allows up to 80 units at any given time, and orders can be placed in increments of no more than 30 units. Could you assist me in developing an optimal inventory strategy considering these parameters?



**LLM
Consultant**

What planning time horizon (in days) should we optimize the inventory strategy for? For example, 30 days, 90 days, or 365 days.



**Human
Imitator**

Let's assume a 90-day planning period. This aligns well with our seasonal sales patterns and accounts for potential fluctuations over the quarter.



**LLM
Consultant**

Last question, about risk preference, on a scale from -10 (very risk-averse) to 10 (very risk-seeking), how would you rate your business strategy?



**Human
Imitator**

On this scale, my approach leans towards moderate risk aversion. A score of 3 reflects a cautious stance where stability and reliability are prioritized over aggressive growth strategies. This rating ensures that we maintain sufficient stock levels to satisfy customer needs while minimizing unnecessary inventory buildup.

Example 2 - Treatment B



**Human
Imitator**

Hi, I manage an online electronics store selling phone chargers. We have unpredictable daily demand that follows a Poisson distribution with $\lambda=6$ units/day. If we don't meet demand, customers will leave without purchasing anything. Holding costs us \$0.50/unit/day, while each unfulfilled order incurs a penalty of \$2.75/unit/day. There's also a fixed cost of \$3.00 per shipment and a 4-day delivery delay. Our storage capacity allows up to 80 units at any given time, and orders can be placed in increments of no more than 30 units. Could you assist me in developing an optimal inventory strategy considering these parameters?

Great, thanks for the clarification. Given the information provided, we'll focus on creating an inventory policy that minimizes costs while ensuring demand is met as efficiently as possible. Here's how we'll proceed:

1. **Determine Reorder Point and Order Quantity:**
 - We'll calculate the reorder point (R) considering the lead time and average demand.
 - We'll decide on an order quantity (Q) that balances holding costs, ordering costs, and stockout costs.

2. **Consideration of Lead Time:**
 - With a lead time of 4 days and an average demand of 6 units/day, we expect demand during lead time to be 4 days * 6 units/day = 24 units.

3. **Reorder Point (R):**
 - To account for variability in demand and prevent stockouts, we can use a safety stock approach. Typically, safety stock is calculated as a function of demand variability and lead time, but since we have a Poisson distribution, we will consider a slightly higher level of safety stock to cover unexpected demand.
 - Let's use a safety stock of 10 units, giving us a reorder point of $(R = 24 + 10 = 34)$ units.

4. **Order Quantity (Q):**
 - Since your maximum order increment is 30 units, and considering your storage capacity, we'll aim to order up to your maximum capacity when inventory falls below the reorder point.
 - Hence, you should order enough units to bring the total inventory up to 80 units, the storage capacity.

Inventory Policy Recommendation:
 - **Reorder Policy:** When the sum of on-hand and on-order inventory falls below 34 units, place an order.
 - **Order Size:** Order enough units to bring the total inventory level up to 80 units, with a maximum order size of 30 units per order due to constraints.

This policy aims to balance the cost of holding inventory and the risk of stockouts while adhering to your storage and order size constraints. Would you like to discuss any aspect of this strategy further or explore different scenarios?



**LLM
Consultant**

Certainly! Let's break down the information you've provided and gather any additional details needed to create an optimal inventory strategy for your phone chargers.

1. **Demand Characteristics:**
 - Daily demand follows a Poisson distribution with $\lambda = 6$ units/day.

2. **Costs:**
 - Holding cost: \$0.50 per unit per day.
 - Stockout penalty: \$2.75 per unit per day.
 - Fixed ordering cost: \$3.00 per shipment.

3. **Lead Time:**
 - 4-day delivery delay for orders.

4. **Storage and Order Constraints:**
 - Maximum storage capacity: 80 units.
 - Maximum order size: 30 units.

To start, could you confirm if there are any other costs or constraints we should consider? Additionally, do you have any preferences or limitations regarding how often orders can be placed (e.g., daily, weekly)?



**Human
Imitator**

No other costs or constraints besides what was mentioned earlier. Orders can be placed as needed based on inventory levels and demand forecasts.

Figure 13 Complete conversation logs for Example 2 (continued on next page).

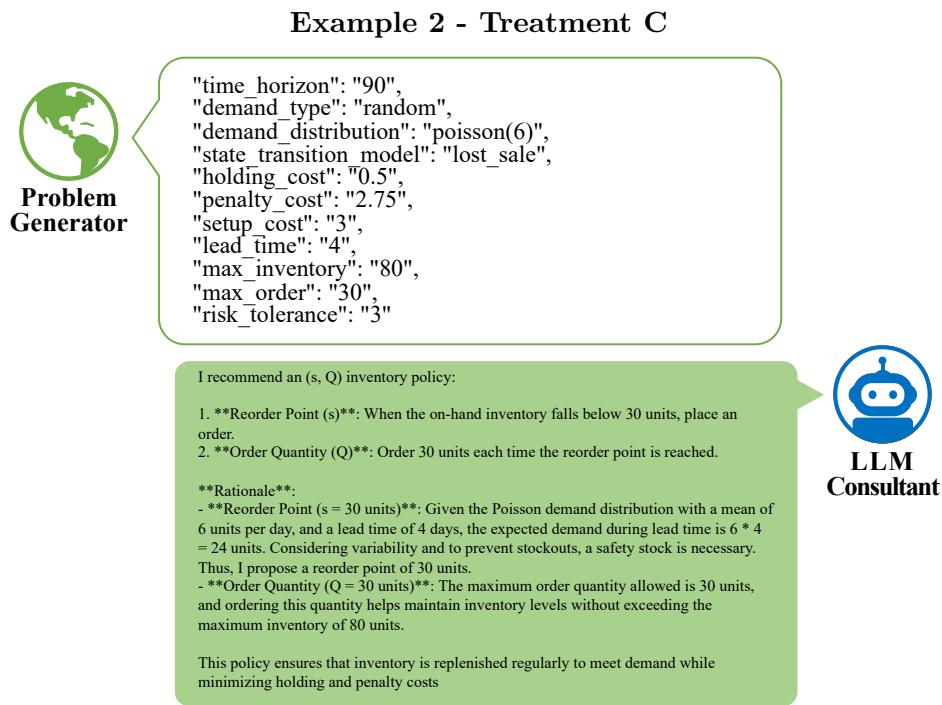


Figure 13 Complete conversation logs for Example 2 (continued).

Example 3

Quick clarification before I fill the distribution: you said 'weekly demand follows a normal distribution with mean 12 and sd 3' and asked to use daily units. Which conversion do you want me to use for daily demand? Reply with either:

A) Model daily demand as independent daily normals such that the weekly sum is $N(12,3)$. This implies daily mean = $12/7$ and daily sd = $3/\sqrt{7}$.

B) Simply scale the weekly mean and sd by $1/7$, i.e., daily mean = $12/7$ and daily sd = $3/7$.

Also, for the inventory dynamics, should unmet demand be treated as 'lost_sale' or as 'backlog'? Please answer with your choice for the conversion (A or B) and exactly one of 'lost_sale' or 'backlog'.

(Notice: I will ask about holding/penalty/setup costs, lead time, capacities, and finally risk tolerance — I will ask the risk-tolerance question last.)



Human Imitator

A) Model daily demand as independent daily normals such that the weekly sum is $N(12,3)$. This implies daily mean = $12/7$ and daily sd = $3/\sqrt{7}$.



LLM Consultant

Example 4

You previously indicated unmet demand results in customers leaving (state_transition_model = 'lost_sale'). You have also stated the penalty cost is \$2.80 per unit per day for unmet demand. These two statements conflict: in a lost-sale model unmet demand is a one-time lost sale (so a one-time penalty per lost unit would be appropriate), whereas a per-unit-per-day penalty implies unmet demand is backlogged and accrues daily (state_transition_model = 'backlog'). Please clarify which of the two you intend: (A) Keep state_transition_model = 'lost_sale' and confirm the penalty is actually \$2.80 one-time per lost unit (i.e., record unit as "\$/unit"), or (B) You want to model backorders so unmet demand accrues a penalty of \$2.80 per unit per day (in which case please confirm changing state_transition_model to 'backlog' so the per-day penalty is meaningful). If you intend a different formulation, please specify precisely.



Human Imitator

Keep state_transition_model = 'lost_sale'. Confirm the penalty is actually \$2.80 one-time per lost unit.



LLM Consultant

Example 5

I detected a conflict: the table currently has time_horizon = 30 days, but you stated the planning time horizon is 90 days. Which value should I use for the planning time horizon (30 or 90 days)? Please confirm if I should update the table to 90 days.



Human Imitator

Use 90 days for the planning time horizon.



LLM Consultant

Figure 14 Excerpts of representative conversations (Examples 3, 4, and 5) demonstrating conflict resolution and clarification.