Deep Learning for Sequential Decision Making under Uncertainty: Foundations, Frameworks, and Frontiers

İ. Esra Büyüktahtakın
Grado Department of Industrial and Systems Engineering, Virginia Tech
Blacksburg, VA 24061, USA
esratoy@vt.edu

April 12, 2026
Preprint

Abstract

Artificial intelligence (AI) is moving increasingly beyond prediction to support decisions in complex, uncertain, and dynamic environments. This shift creates a natural intersection with operations research and management sciences (OR/MS), which have long offered conceptual and methodological foundations for sequential decision-making under uncertainty. At the same time, recent advances in deep learning, including feedforward neural networks, LSTMs, transformers, and deep reinforcement learning, have expanded the scope of data-driven modeling and opened new possibilities for large-scale decision systems. This tutorial presents an OR/MS-centered perspective on deep learning for sequential decision-making under uncertainty. Its central premise is that deep learning is valuable not as a replacement for optimization, but as a complement to it. Deep learning brings adaptability and scalable approximation, whereas OR/MS provides the structural rigor needed to represent constraints, recourse, and uncertainty. The tutorial reviews key decision-making foundations, connects them to the major neural architectures in modern AI, and discusses leading approaches to integrating learning and optimization. It also highlights emerging impact in domains such as supply chains, healthcare and epidemic response, agriculture, energy, and autonomous operations. More broadly, it frames these developments as part of a wider transition from predictive AI toward decision-capable AI and highlights the role of OR/MS in shaping the next generation of integrated learning–optimization systems.

Keywords: Sequential decision-making, Deep learning, Operations research and management science, Optimization, Stochastic programming, Reinforcement learning, Dynamic programming, Transformers, Large language models, Predict-then-optimize, Learning-to-optimize (L2O), Decision-focused learning

1 Introduction and Motivation

Artificial intelligence (AI) is increasingly moving beyond pattern recognition and prediction toward systems that support, recommend, and sometimes automate decisions. In many domains central to operations research

and management science (OR/MS)—including healthcare, supply chains, energy, finance, transportation, and public policy—the fundamental challenge is not only to anticipate what may happen, but to determine what should be done as uncertainty unfolds over time. Sequential decision-making under uncertainty thus stands as one of the most consequential points of contact between modern AI and OR/MS.

Recent advances in deep learning have greatly expanded the reach of data-driven modeling. Feedforward neural networks (FNNs) have proven effective for nonlinear approximation in static settings, recurrent architectures such as long short-term memory networks (LSTMs) have strengthened learning in temporal environments [57, 61], and transformer architectures now underpin large language models, generative AI, and conversational systems such as ChatGPT [121]. Deep reinforcement learning has further broadened the ability to learn sequential decision policies through interaction in complex dynamic settings [117]. Together, these developments have substantially increased both the representational power and the practical scope of modern learning systems.

At the same time, their growing use in operational settings has made a longstanding distinction impossible to ignore: prediction is not the same as decision-making. A highly accurate predictive model does not necessarily produce a high-quality decision, especially when actions are constrained, uncertainty evolves over time, and downstream consequences matter [46]. In sequential settings, the distinction becomes sharper still, because present decisions shape future opportunities, risks, and feasible actions. The relevant objective is, therefore, rarely prediction alone, but the design of policies that remain feasible, adaptive, and effective over time. This is where OR/MS provides an essential perspective and methodology. The field has long developed rigorous frameworks for sequential decision-making under uncertainty, including dynamic programming, Markov decision processes, stochastic programming, approximate dynamic programming, and reinforcement learning [12, 23, 104, 113, 117]. These frameworks place decisions at the center of the analysis by explicitly accounting for constraints, recourse, uncertainty, information structure, and system-level trade-offs. They ask not merely whether a model predicts well, but whether a policy is implementable, respects what is known at each stage, and improves operational performance.

However, classical decision-making frameworks face serious challenges in contemporary settings. Many sequential problems involve large state spaces, long planning horizons, heterogeneous data streams, complex constraints, and mixed-integer structure, rendering exact dynamic programming intractable and large-scale stochastic programs computationally burdensome. Deep learning brings important strengths to this landscape: neural architectures can approximate complex functional relationships, learn compact representations from large datasets, and in some settings generalize across families of problem instances rather than treating each instance in isolation [16]. The central question then is not whether learning should replace optimization, but how learning and optimization can be integrated so that each contributes what it does best.

One useful lens for this integration is the *predict-then-optimize* paradigm, in which uncertain inputs are first predicted and then used in a downstream optimization model. In OR/MS, this perspective has been sharpened by work showing that prediction error and decision error are not the same, and that learning should often be aligned with downstream optimization quality rather than prediction accuracy alone [46]. In parallel, the AI community has developed closely related task-based and end-to-end learning approaches that train models directly against the eventual stochastic optimization objective [42]. These developments

point to a broader shift: deep learning architectures should be studied not only as predictive tools but also as components of structured decision systems.

This tutorial treats deep learning architectures not merely as predictive tools, but as components of structured decision systems for sequential decision-making under uncertainty. It focuses on how these architectures can be integrated with OR/MS frameworks in ways that preserve feasibility, non-anticipativity, recourse, and decision quality. The central questions are what should be learned in structured sequential settings, when reinforcement learning is more appropriate than stochastic programming, and how newer architectures such as transformers and large language models can be incorporated without sacrificing the discipline imposed by constraints, objectives, and information structure. More broadly, the tutorial argues that the next phase of AI for high-impact operations will depend not only on more powerful learning architectures but also on stronger decision-making frameworks.

Tutorial Overview. The remainder of this tutorial is organized as follows. Section 2 reviews the foundations of sequential decision-making under uncertainty, including Markov decision processes, dynamic programming, reinforcement learning, and multi-stage stochastic programming. Section 3 introduces the major deep learning architectures relevant to decision systems, with emphasis on FNNs, graph neural networks, LSTMs, and transformer/LLM architectures. Section 4 discusses key approaches for combining learning and optimization, including predict-then-optimize and decision-aware learning. Section 5 focuses on constrained and multi-stage decision learning, with particular attention to feasibility and non-anticipativity. Section 6 examines deep reinforcement learning for sequential and combinatorial settings. Section 7 highlights representative applications and broader interdisciplinary impact. Section 8 discusses open challenges and research frontiers. Finally, Section 9 concludes the tutorial.

2 Preliminaries: Sequential Decision Making under Uncertainty

2.1 Sequential Decision-Making Foundations

Sequential decision-making under uncertainty is a foundational theme in operations research, since decisions are rarely made all at once and instead must adapt over time as information is progressively revealed [12, 23, 104, 113]. We consider a finite horizon $t = 1, \ldots, T$ and a stochastic process $\{\xi_t\}_{t=1}^{T}$, where $\xi_{[t]} := (\xi_1, \ldots, \xi_t)$ denotes the information available by stage t. At each stage, the decision is chosen after observing the current history, so $x_t = x_t(\xi_{[t]})$. Thus, the information pattern is

$$\begin{aligned} &\mathit{decide}(x_1) \;\rightarrow\; \mathit{observe}(\xi_2) \;\rightarrow\; \mathit{decide}(x_2) \;\rightarrow\; \cdots \\ &\cdots \;\rightarrow\; \mathit{observe}(\xi_T) \;\rightarrow\; \mathit{decide}(x_T). \end{aligned}$$

Accordingly, each x_t may depend only on $\xi_{[t]}$, not on future realizations; this is the *non-anticipativity* principle [23, 113]. To allow both continuous and integer decisions, let $x_t(\xi_{[t]}) \in \mathcal{X}_t(\xi_{[t]}) \subseteq \mathbb{R}_+^{n_t - q_t} \times \mathbb{Z}_+^{q_t}$. The goal is to minimize the expected total cost over the horizon, where the stage-t term $c_t(\xi_{[t]})^\top x_t(\xi_{[t]})$ represents the immediate cost incurred after observing $\xi_{[t]}$, and later decisions provide recourse as uncertainty

unfolds. Suppressing the explicit dependence on $\xi_{[t]}$ for readability, a compact multi-stage formulation is

$$\begin{aligned} \min_{x} \quad & \mathbb{E}\left[\sum_{t=1}^{T} c_t^\top x_t\right] \\ \text{s.t.} \quad & H_1 x_1 = b_1, \\ & A_t x_{t-1} + H_t x_t = b_t, \qquad t = 2, \dots, T, \\ & x_t \in \mathcal{X}_t, \qquad t = 1, \dots, T. \end{aligned} \tag{1}$$

For expositional simplicity, (1) is written in a linear recourse form, but the same sequential structure extends more broadly to problems with nonlinear objectives and constraints. Here, for $t \geq 2$, the quantities A_t, H_t, b_t, c_t, and $\mathcal{X}_t$ depend on the realized history $\xi_{[t]}$. In practice, the uncertainty process is often approximated by a finite scenario tree, where each node corresponds to a realized history and non-anticipativity requires identical decisions at nodes sharing the same history [23, 113]. This formulation provides a common foundation for MDPs, dynamic programming, reinforcement learning, and multi-stage stochastic programming, as summarized in Figure 1.

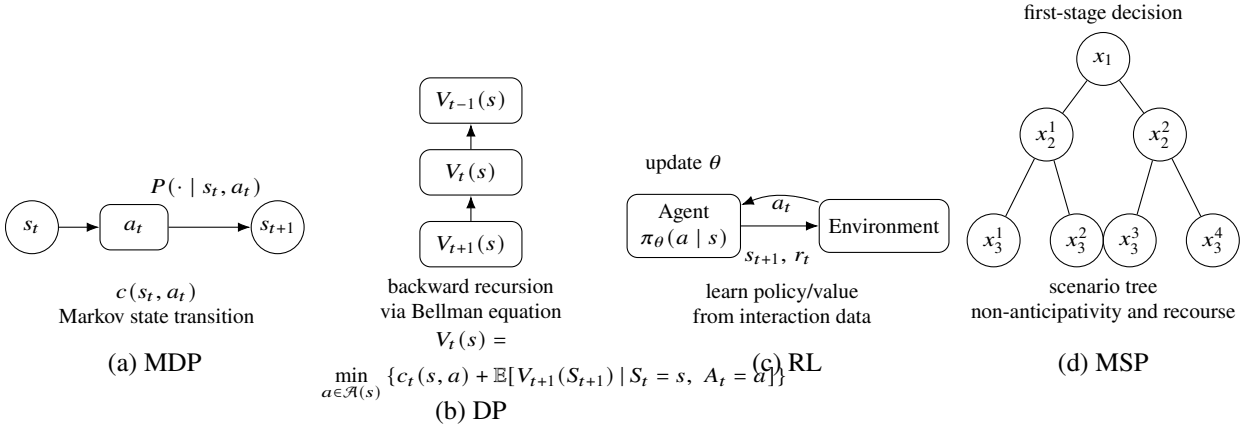


Figure 1: Comparison of four sequential decision-making frameworks. (a) A Markov decision process (MDP) models stochastic state transitions under actions and stage costs. (b) Dynamic programming (DP) solves such problems through Bellman recursion over value functions. (c) Reinforcement learning (RL) learns policies or value functions from interaction with an environment. (d) Multi-stage stochastic programming (MSP) represents decisions explicitly on a scenario tree with recourse and non-anticipativity constraints.

2.2 Markov Decision Processes

The formulation above is history-based: at stage t, the decision $x_t = x_t(\xi_{[t]})$ may depend on all information revealed so far. In many settings, however, the decision-relevant information in the history can be summarized by a state variable s_t, leading to the framework of a Markov decision process (MDP) [18, 104]. An MDP is defined by $(\mathcal{S}, \mathcal{A}, P, c, \gamma, \rho_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}(s)$ is the set of feasible actions in state s, $P(s' \mid s, a)$ is the transition probability, $c(s, a)$ is the one-stage cost, $\gamma \in (0, 1]$ is a discount factor, and ρ_0 is the initial state distribution. At stage t, the decision-maker observes s_t, chooses an action $a_t \in \mathcal{A}(s_t)$, the

system evolves as $s_{t+1} \sim P(\cdot \mid s_t, a_t)$, and incurs cost $c(s_t, a_t)$.

Relative to the notation in the previous subsection, the action a_t plays the role of the stage decision, while the state s_t summarizes the decision-relevant information contained in the history $\xi_{[t]}$. More precisely, when there exists a mapping $s_t = \phi_t(\xi_{[t]})$ such that the feasible actions, one-stage costs, and conditional distribution of the next state depend on the history only through s_t, one may restrict attention to policies of the form $a_t = \pi_t(s_t)$ without loss of optimality [18, 104]. The key assumption is the Markov property:

$$\mathbb{P}(s_{t+1} \mid s_1, a_1, \ldots, s_t, a_t) = \mathbb{P}(s_{t+1} \mid s_t, a_t),$$

which states that, given the current state and action, the next state is independent of the earlier history.

The objective is to find a policy π mapping states to actions so as to minimize expected cumulative discounted cost:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=1}^{T} \gamma^{t-1} c(s_t, a_t) \right]. \tag{2}$$

The corresponding optimal cost-to-go function satisfies the Bellman optimality equation

$$V^*(s) = \min_{a \in \mathcal{A}(s)} \Big\{ c(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a)\, V^*(s') \Big\}. \tag{3}$$

Thus, an MDP may be viewed as a special case of the general sequential model when the decision-relevant history can be represented by a Markov state without loss of optimality. This abstraction is powerful and computationally convenient, but it can be restrictive in operations research applications with complex feasibility constraints, integer decisions, and explicit scenario-based coupling, which are often represented more naturally in optimization-based formulations.

2.3 Dynamic Programming

Given the MDP formulation above, dynamic programming (DP) provides a recursive framework for characterizing optimal sequential decisions through value functions [12, 18]. Let $V_t(s)$ denote the optimal cost-to-go when the system is in state s at stage t. Then the Bellman recursion is

$$\begin{aligned} V_t(s) = \min_{a \in \mathcal{A}(s)} \Big\{ & c_t(s,a) \\ & + \mathbb{E}[V_{t+1}(s_{t+1}) \mid s_t = s,\ a_t = a] \Big\}, \end{aligned} \tag{4}$$

with terminal condition $V_{T+1}(\cdot) = 0$.

This recursion captures the central logic of sequential decision-making: the optimal value at the current stage equals the immediate cost plus the expected future cost induced by the current action. In this sense, DP provides the main analytical bridge between the state-based MDP representation and optimal decision-making over time. Its main limitation is computational. As the state space grows in dimension or granularity, exact evaluation and storage of the value functions quickly become intractable, leading to the well-known curse of dimensionality [12]. This challenge has motivated approximate dynamic programming, reinforcement

learning, and other learning-based methods designed to scale sequential decision-making to high-dimensional settings [19, 101, 117].

2.4 Reinforcement Learning

Reinforcement learning (RL) builds on the same MDP framework, but addresses settings in which the transition law, the one-stage cost, or both are not known explicitly and must instead be learned from data or interaction with the environment [77, 117, 125]. A policy $\pi(a \mid s)$ specifies how actions are selected in each state, and the goal is to learn a policy that minimizes expected cumulative cost.

Broadly speaking, RL methods can be grouped into three main classes. Value-based methods estimate quantities such as the value function $V_t(s)$ or the action-value function $Q_t(s, a)$ and derive decisions from these estimates. Policy-based methods instead parameterize the policy directly and optimize it from data. Actor–critic methods combine these two ideas by learning both a policy representation and a value-based performance signal.

Relative to classical DP, RL replaces exact model-based recursion with data-driven approximation, making it attractive for high-dimensional problems and environments that are difficult to model analytically. At the same time, standard RL formulations do not naturally enforce the rich feasibility constraints, integrality requirements, and explicit non-anticipativity conditions that arise in many operations research applications. Thus, while RL offers scalable tools for learning sequential decisions, additional modeling structure is often needed in constrained OR settings. In many contemporary applications, these ideas are often combined with deep neural networks to approximate value functions or policies, giving rise to deep reinforcement learning, which we discuss in later sections.

2.5 Multi-Stage Stochastic Programming

While MDPs, DP, and RL adopt a state-based view, multi-stage stochastic programming (MSP) remains closer to the original history-based formulation in (1), where decisions adapt to revealed information and are linked across stages through recourse constraints [23, 113]. In MSP, this structure is represented explicitly over a finite set of scenarios, each corresponding to a realization of the uncertainty process.

In a scenario-based extensive formulation, stage-wise decisions are indexed by both time and scenario, say x_t^ω, and the objective minimizes expected total cost, typically of the form $\sum_{\omega\in\Omega} p^\omega \sum_{t=1}^{T} c_t^{\omega\top} x_t^\omega$, subject to scenario-wise feasibility and non-anticipativity. The latter requires $x_t^\omega = x_t^{\omega'}$ whenever scenarios ω and ω' share the same history up to stage t. Figure 1(d) illustrates this scenario-tree view, where branching captures uncertainty revelation and downstream decisions provide recourse.

Compared with the MDP framework, MSP places greater emphasis on explicit decision variables and constraint systems. This is especially valuable in OR/MS applications involving network flows, capacity limits, logical conditions, budget restrictions, or mixed-integer decisions, where structural fidelity is often more important than a compact state representation. The tradeoff is computational: scenario-based formulations grow rapidly with the number of stages and uncertainty realizations, and integer decisions make the resulting extensive forms even more challenging, motivating decomposition and scenario-based acceleration

methods such as stage-t scenario dominance [23, 29].

2.6 Perspective and Implications

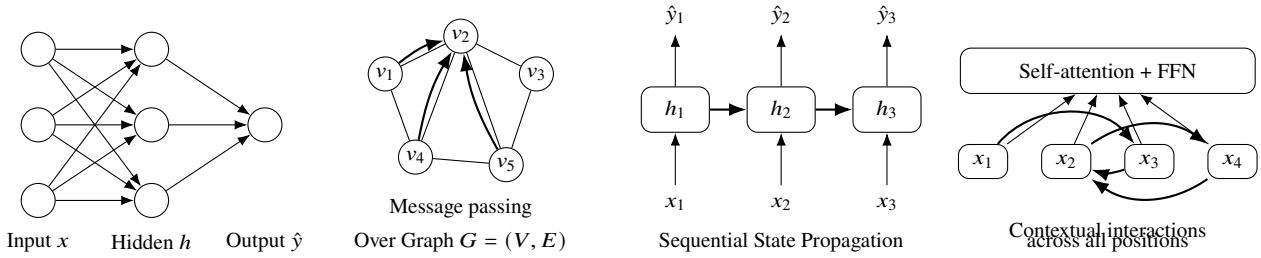
The frameworks above are best viewed as complementary ways of representing sequential decision-making under uncertainty. The general formulation in (1) begins from the broadest view: uncertainty is revealed over time, decisions adapt as information arrives, and recourse and non-anticipativity are explicit. MDPs impose additional structure by summarizing decision-relevant history through a state variable; dynamic programming exploits that structure through Bellman recursion; reinforcement learning replaces explicit model knowledge with data-driven estimation of value functions or policies; and multi-stage stochastic programming stays closest to the original history-based view by modeling decisions and constraints explicitly across scenarios.

From an OR/MS perspective, the main difference among these frameworks lies in how they balance modeling fidelity and computational tractability. MDPs and DP offer analytical clarity and strong optimality principles, but depend on a manageable state representation. RL scales more naturally to complex and high-dimensional settings, yet standard formulations do not directly enforce feasibility, integrality, or non-anticipativity. MSP captures these features explicitly and is therefore especially valuable in constrained and multi-scale OR settings, though often at a substantial computational cost.

This tension between structure and scalability helps explain the growing interest in integrating learning and optimization. The central question is not whether one should replace the other, but how their complementary strengths can be combined. That, in turn, raises a natural next question: which neural architectures are best suited to represent sequential decisions, value functions, and policies in complex OR/MS settings?

3 Neural Deep Learning Architectures from a Decision-Making Lens

This section summarizes five core neural architectures used in learning-based decision systems: feedforward neural networks (FNNs), graph neural networks (GNNs), recurrent neural networks (RNNs), long- and short-term memory networks (LSTMs) and transformer/LLM architectures. These architectures build on a broad neural learning foundation established by early work on multilayer neural networks and representation learning [57, 82, 108]. Using a unified notation, we highlight how they progressively capture richer forms of structure relevant to optimization and decision-making, including static feature mappings, relational dependencies, sequential dynamics, and long-range contextual interactions. In particular, FNNs provide the classical foundation for static nonlinear approximation, GNNs extend learning to graph-structured domains through message passing [58, 75, 112], RNNs and LSTMs model sequential dependence through recurrent state transitions and gated memory [47, 61], and transformers enable context-dependent representation learning through self-attention [121]. Throughout, x denotes an input, h a hidden representation, $\hat{y}$ a model output or predicted decision, and θ the full set of trainable parameters. Figure 2 provides a visual comparison of the four neural architectures discussed in this section and highlights the main structural mechanism underlying each model.



(a) FNN (b) GNN (c) RNN/LSTM (d) Transformer/LLM

Figure 2: Comparison of FNN, GNN, RNN/LSTM, and transformer architectures for learning-based decision systems.

3.1 Feedforward Neural Networks (FNNs)

Feedforward neural networks are the canonical architecture for *static* decision mappings. They are most appropriate when the output depends only on the current feature vector, rather than on temporal history, graph structure, or broader contextual interactions. This makes them a natural starting point for predict-then-optimize pipelines, surrogate optimization, and approximation of static decision rules [57, 62, 108].

Let $x \in \mathbb{R}^{d_0}$ be the input. An L-layer feedforward neural network defines hidden representations $h^{(1)}, \ldots, h^{(L-1)}$ and output $\hat{y} \in \mathbb{R}^{d_L}$ through

$$\begin{aligned} h^{(0)} &= x, \\ h^{(\ell+1)} &= \sigma^{(\ell)}\left(W^{(\ell)} h^{(\ell)} + b^{(\ell)}\right), \end{aligned} \tag{5}$$

for $\ell = 0, \ldots, L-1$, with

$$\hat{y} = h^{(L)}. \tag{6}$$

Here, $h^{(\ell)} \in \mathbb{R}^{d_\ell}$, $W^{(\ell)} \in \mathbb{R}^{d_{\ell+1} \times d_\ell}$, $b^{(\ell)} \in \mathbb{R}^{d_{\ell+1}}$, and $\sigma^{(\ell)}(\cdot)$ is a nonlinear activation. The parameter set is

$$\theta = \{(W^{(0)}, b^{(0)}), \ldots, (W^{(L-1)}, b^{(L-1)})\}.$$

The key mechanism is repeated nonlinear feature transformation: each layer applies an affine map followed by a nonlinearity, allowing the network to construct progressively richer latent representations. The resulting model implements a nonlinear function $f_\theta : \mathbb{R}^{d_0} \to \mathbb{R}^{d_L}$, which explains the expressive power of FNNs and their classical role as universal approximators [57, 62]. As illustrated in Figure 2(a), an FNN maps input features to outputs through a sequence of layered nonlinear transformations.

In OR, FNNs are useful for predicting optimal or near-optimal decisions, approximating objective values or recourse functions, learning dispatching or ranking rules, and embedding fast surrogate models inside larger optimization pipelines. Given training data $\{(x_i, y_i)\}_{i=1}^N$, the network is typically learned by minimizing

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(x_i), y_i), \tag{7}$$

where $\ell(\cdot, \cdot)$ is an application-specific loss. Training is commonly carried out by gradient-based optimization

using backpropagation [57, 108].

FNNs are flexible and computationally efficient, but they are memoryless: they do not naturally encode sequential dependence, relational structure, or long-range contextual interactions. These limitations motivate richer architectures such as GNNs, recurrent models, and transformers [57].

3.2 Graph Neural Networks (GNNs)

Many OR/MS problems are naturally defined on graphs, such as transportation networks, supply chains, routing systems, and precedence structures. In these settings, the input is not simply a feature vector, but a graph $G = (V, E)$ in which nodes and edges encode relationships among entities. Graph Neural Networks (GNNs) are designed for such data by learning representations that combine node features with graph structure, making them especially well suited for optimization problems in which connectivity and local interactions play a central role [58, 75, 112, 132].

Let $h_v^{(\ell)}$ denote the representation of node $v \in V$ at layer ℓ, with $h_v^{(0)}$ as its initial feature vector. A generic GNN layer updates each node by combining its current representation with aggregated information from its neighbors:

$$h_v^{(\ell+1)} = \psi^{(\ell)}\Big(h_v^{(\ell)}, \mathrm{AGG}\big(\{\, h_u^{(\ell)} : u \in \mathcal{N}(v) \,\}\big)\Big), \tag{8}$$

where $\mathcal{N}(v)$ is the set of neighbors of node v, $\mathrm{AGG}(\cdot)$ is a permutation-invariant aggregation operator such as sum, mean, or max, and $\psi^{(\ell)}(\cdot)$ is a learnable update function. The key architectural idea is that each node repeatedly exchanges information with its local neighborhood. After several layers, the representation of node v reflects not only its own features, but also structural information propagated from its multi-hop neighborhood.

This mechanism is commonly referred to as *message passing*. At each layer, a node gathers information from adjacent nodes, aggregates it into a summary, and updates its own embedding. Repeating this process enables the network to learn representations that capture both local attributes and relational structure. As illustrated in Figure 2(b), a GNN therefore extends neural learning from flat feature vectors to graph-structured domains, where the relationships among entities are often as important as the entities themselves.

For graph-level prediction, the final node representations are combined into a graph representation

$$h_G = \rho\big(\{h_v^{(L)} : v \in V\}\big), \tag{9}$$

where $\rho(\cdot)$ is a permutation-invariant pooling operator, such as sum, mean, or max. The graph representation is then mapped to the final prediction,

$$\hat{y} = f_\theta(h_G). \tag{10}$$

Thus, depending on the task, a GNN can produce either node-level outputs or a graph-level output.

For OR and decision-making, GNNs are especially useful when solution quality depends on network topology, neighborhood effects, or interactions among entities. Typical applications include routing, facility location, network design, scheduling with precedence relations, and learning heuristics for combinatorial optimization. Relative to feedforward neural networks, which operate on fixed-dimensional feature vec-

tors, GNNs explicitly exploit relational structure and are therefore better aligned with many classical OR formulations [16, 132].

3.3 Sequential Architectures: RNNs, LSTMs, and Temporal Convolutions

Many OR problems are inherently sequential: the current decision depends not only on the current input, but also on past states, observations, or actions. Recurrent neural networks (RNNs) are designed for this setting by maintaining a hidden state that evolves over time and summarizes relevant history [47, 57, 108].

Let $x_{1:T} = (x_1, \dots, x_T)$ be a sequence, where $x_t \in \mathbb{R}^{d_x}$. An RNN maintains a hidden state $h_t \in \mathbb{R}^{d_h}$ and output $\hat{y}_t \in \mathbb{R}^{d_y}$ through

$$\begin{aligned} h_t &= \phi_\theta(x_t, h_{t-1}), \\ \hat{y}_t &= \psi_\theta(h_t), \qquad t = 1, \dots, T, \end{aligned} \tag{11}$$

where h_0 is initialized in advance. A standard parametrization is

$$\begin{aligned} h_t &= \varphi(W_x x_t + W_h h_{t-1} + b_h), \\ \hat{y}_t &= \rho(W_y h_t + b_y). \end{aligned} \tag{12}$$

The key architectural idea is recurrence: the same transition map is reused across time, so the model updates its internal state as new inputs arrive. As illustrated in Figure 2(c), this shifts the architecture from a static mapping $x \mapsto \hat{y}$ to a dynamic mapping $(x_t, h_{t-1}) \mapsto (h_t, \hat{y}_t)$.

For OR/MS readers, the main appeal is clear: recurrent models can encode evolving system state, making them useful for applications such as inventory control, routing under changing information, time-dependent scheduling, and iterative improvement processes. Given targets $y_{1:T}$, training typically minimizes

$$\mathcal{L}(\theta) = \frac{1}{T} \sum_{t=1}^{T} \ell(\hat{y}_t, y_t), \tag{13}$$

using backpropagation through time [57, 127]. However, standard RNNs often struggle with long-range dependencies because gradients can vanish or explode over many recurrent steps [17].

Long short-term memory (LSTM) networks address this limitation by introducing an explicit memory cell c_t and gates that regulate information retention, update, and exposure [55, 61]. Given (x_t, h_{t-1}, c_{t-1}),

an LSTM computes

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \tag{14}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \tag{15}$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c), \tag{16}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \tag{17}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \tag{18}$$

$$h_t = o_t \odot \tanh(c_t), \tag{19}$$

where f_t, i_t, and o_t are the forget, input, and output gates. The main architectural advantage is that the LSTM separates memory storage from hidden-state output: it can preserve important information over longer horizons while selectively updating and revealing that memory. The forget gate controls what past information is retained, the input gate controls what new information is written, and the output gate determines what portion of the updated cell state is exposed through h_t. This makes LSTMs substantially more effective than vanilla RNNs in long-horizon or delayed-effect settings [57, 61].

Convolutional architectures, such as temporal convolutional neural networks (TCNNs), can also be used for sequential data by applying one-dimensional filters across time to extract local temporal patterns. With causal and dilated convolutions, they can expand the receptive field without recurrence, providing a parallelizable alternative to RNN- and LSTM-based sequence modeling [10, 83, 120]. They can be particularly useful when high-quality decisions depend on detecting local and multi-scale temporal structure, such as recurring demand fluctuations, evolving system states, or short-horizon operational signals.

For decision-making, recurrent architectures are particularly useful when decisions depend on evolving latent state rather than only on the current feature vector. Their main strength is history dependence; while their main limitation is that information must still be processed sequentially, which restricts parallelization and makes very long-context reasoning difficult. These limitations motivate attention-based architectures discussed in the next section.

3.4 Transformer/LLM Architectures and Attention

Transformer architectures are a central framework for modern sequence modeling and form the basis of most large language models (LLMs) [57, 121]. Their defining innovation is the attention mechanism, which allows each element of a sequence to directly incorporate information from other relevant elements rather than relying on a recurrent hidden state. This makes transformers especially useful when decisions depend on long-range interactions, variable-length context, or rich dependencies across multiple inputs.

Let $x_{1:n} = (x_1, \ldots, x_n)$ denote an input sequence. Each element x_i is first mapped to an embedding $e_i \in \mathbb{R}^d$. Since transformers do not inherently encode order, a positional encoding $p_i \in \mathbb{R}^d$ is added, giving

the initial representation

$$h_i^{(0)} = e_i + p_i, \qquad i = 1, \ldots, n. \tag{20}$$

Thus, each input representation contains both content and position information.

The core operation is *self-attention*. At layer ℓ, let $H^{(\ell)} \in \mathbb{R}^{n \times d}$ denote the matrix of sequence representations. The model forms three learned projections:

$$Q = H^{(\ell)} W_Q, \qquad K = H^{(\ell)} W_K, \qquad V = H^{(\ell)} W_V, \tag{21}$$

where Q, K, and V are the query, key, and value matrices. Attention is then computed as

$$\mathrm{Attn}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V. \tag{22}$$

The main insight is straightforward: each position compares itself with all other positions, assigns larger weight to the most relevant ones, and updates its representation by taking a weighted combination of their value vectors. In contrast to recurrent models, information does not need to travel step by step through time. As illustrated in Figure 3, each element can interact directly with any other relevant element in the sequence, making it easier to capture long-range dependencies [121].

Figure 3 summarizes the self-attention mechanism introduced above, showing how relevance scores across positions are converted into attention weights and used to form updated contextual representations.

Transformers typically use *multihead attention*, meaning that several attention operations are learned in parallel for each $m = 1, \ldots, M$:

$$\mathrm{head}_m = \mathrm{Attn}(QW_Q^{(m)}, KW_K^{(m)}, VW_V^{(m)}). \tag{23}$$

The resulting heads are then combined. The purpose of multiple heads is to let the model capture different types of relationships at the same time, such as local interactions, long-range dependencies, or structurally important connections [121].

A transformer layer combines multi-head self-attention with a position-wise feedforward network, together with residual connections and normalization. Let

$$S^{(\ell)} = \mathrm{SelfAttn}\left(H^{(\ell)}\right), \tag{24}$$

where $\mathrm{SelfAttn}(\cdot)$ denotes the multi-head self-attention operation. The layer output is then computed as

$$\bar{H}^{(\ell)} = \mathrm{LayerNorm}\left(H^{(\ell)} + S^{(\ell)}\right), \tag{25}$$

$$H^{(\ell+1)} = \mathrm{LayerNorm}\left(\bar{H}^{(\ell)} + \mathrm{FFN}\left(\bar{H}^{(\ell)}\right)\right). \tag{26}$$

Here, self-attention mixes information across positions, while the feedforward network refines each position separately. The residual connections help preserve useful information across layers, and normalization improves training stability [57, 121].

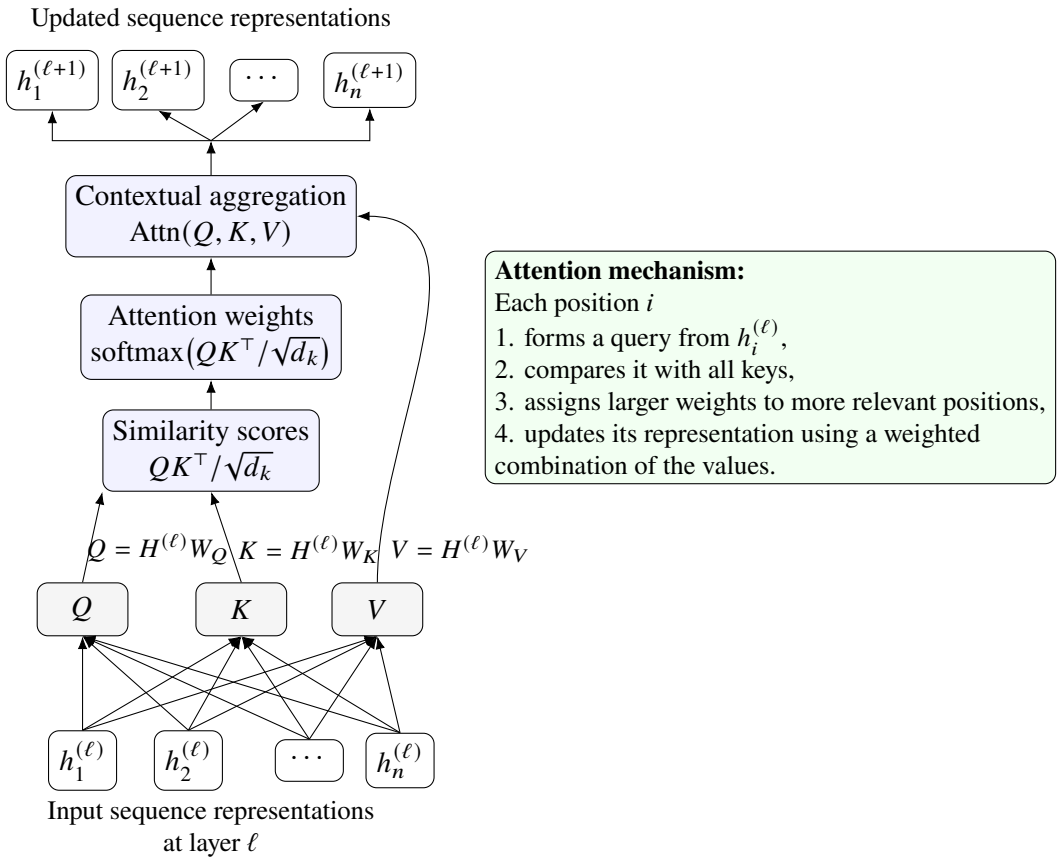


Figure 3: Illustration of the self-attention mechanism. The layer-ℓ token representations $\{h_i^{(\ell)}\}_{i=1}^n$ are stacked into the matrix $H^{(\ell)}$, from which the query, key, and value matrices are formed. Pairwise similarity scores are then computed, normalized into attention weights, and used to produce updated contextual representations $\{h_i^{(\ell+1)}\}_{i=1}^n$. Adapted from Vaswani et al. [121].

Transformers appear in encoder-only, encoder–decoder, and decoder-only forms; encoder-only models learn contextualized representations, encoder–decoder models map an input sequence to an output sequence, and decoder-only models generate tokens autoregressively and form the basis of modern LLMs [25, 121].

For autoregressive sequence generation, the key formulation is

$$P(x_{1:n}) = \prod_{t=1}^{n} P(x_t \mid x_{<t}), \tag{27}$$

which means that the model generates each token conditioned on the preceding context. In decoder-only architectures, this is enforced by causal masking, so each position can attend only to earlier positions. This is the mechanism underlying prompt-based generation in LLMs [25].

For OR/MS decision-making applications, the main advantage of transformers is their ability to model rich contextual interactions without compressing all prior information into a single recurrent state. This is especially useful when decisions depend on long-range dependencies, heterogeneous entities, structured text, or complex constraints embedded in sequential context. Relative to recurrent models, transformers are more parallelizable and often more effective at capturing global and nonlinear structure, making them increasingly

valuable for learning-based optimization and decision support [16, 57].

3.5 Design Considerations for OR/MS Applications

In OR/MS, the value of deep learning depends not only on approximation quality, but on how well it aligns with decision structure. Several considerations are especially important. Feasibility is not automatic, so learned models often need to be paired with optimization layers, constraint-aware architectures, or downstream repair and refinement procedures [2, 5, 138]. Generalization across horizons, scenarios, and problem scales also remains difficult, particularly in sequential settings where small errors may accumulate over time. Interpretability and reliability are equally important in high-stakes applications, where recommendations must be explainable and operationally implementable. At the same time, integrating learned components with optimization models introduces additional modeling and computational trade-offs, while creating new opportunities for decision-aware learning and end-to-end hybrid frameworks [46, 90].

These methods are especially promising when closely related optimization problems must be solved repeatedly under changing inputs. In such settings, learned models can predict high-quality decisions, identify useful solution structure, or provide warm starts, reducing the need to solve every instance from scratch. This theme reappears later through expandable learning–optimization, LSTM–optimization, and non-anticipative learning–optimization frameworks for sequential and multi-stage stochastic problems [136–138].

In OR/MS, deep learning is not a substitute for optimization, but a powerful complement to it. This perspective leads naturally to the next section, which turns to the main ways learning and optimization can be integrated.

4 Learning–Optimization Integration: Paradigms

The integration of learning and optimization has become essential for making high-quality sequential decisions under uncertainty for large-scale problems. As a result, a range of learning–optimization approaches has emerged, differing in how tightly the two are coupled. As illustrated in Figure 4, these approaches range from predict-then-optimize to more integrated learning-to-optimize frameworks that embed decision structure directly into the learning process. This section briefly reviews that landscape. Broader perspectives on this interface are provided by recent survey work on machine learning for combinatorial optimization, end-to-end constrained optimization learning, and contextual optimization under uncertainty, which together help situate the range of approaches discussed here [16, 79, 109].

4.1 Predict-Then-Optimize

The predict-then-optimize paradigm remains the dominant approach in many applications. In this framework, a predictive model is first trained to estimate uncertain parameters ξ, followed by solving an optimization problem:

$$\hat{\xi} = f_\theta(s), \quad x^*(\hat{\xi}) = \arg\min_{x \in \mathcal{X}} c(x, \hat{\xi}). \tag{28}$$

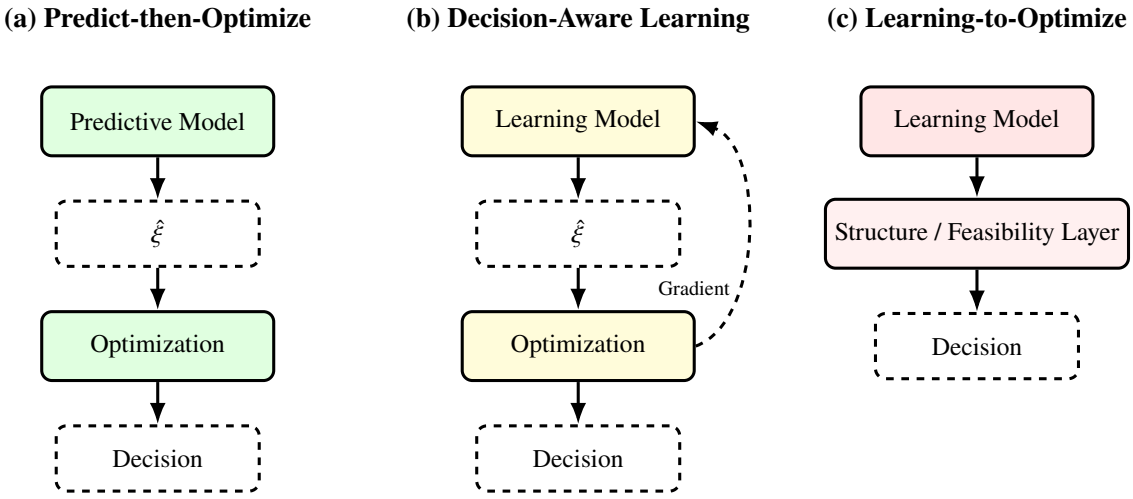


Figure 4: Comparison of learning–optimization paradigms. (a) Predict-then-optimize separates prediction and optimization. (b) Decision-aware learning uses optimization feedback during training, but optimization remains a separate block. (c) Learning-to-optimize more tightly integrates learning and decision generation through a learned or structure-preserving mapping.

This paradigm aligns with classical OR workflows, where forecasts or scenarios are generated and then used within optimization models [23, 113].

Despite its simplicity and modularity, this approach suffers from a fundamental misalignment between predictive accuracy and decision quality. Specifically, predictive models are typically trained by minimizing statistical loss functions such as mean squared error:

$$\min_{\theta} \mathbb{E}\left[\|f_\theta(s) - \xi\|^2\right], \tag{29}$$

which does not necessarily translate into minimizing decision loss:

$$\mathbb{E}\left[c(x^*(f_\theta(s)), \xi)\right]. \tag{30}$$

This issue has been rigorously studied in Elmachtoub and Grigas [46], which shows that small prediction errors can induce large decision errors in structured optimization problems.

Moreover, predict-then-optimize neglects the sensitivity of optimal decisions to prediction errors, a phenomenon closely related to instability in parametric optimization [106]. As a result, this paradigm can yield suboptimal or even infeasible decisions in practice, particularly in high-dimensional or sequential settings.

4.2 Decision-Aware Learning

Decision-aware learning addresses the misalignment between prediction accuracy and decision quality by incorporating the optimization problem directly into the learning objective. Instead of minimizing prediction

error, the model is trained to minimize the expected decision loss:

$$\min_{\theta} \mathbb{E}_{s,\xi} \left[c\left(x^*(f_\theta(s)), \xi\right) \right]. \tag{31}$$

This paradigm often relies on differentiating through the optimization problem or constructing surrogate loss functions that approximate the decision objective. Foundational contributions include Donti et al. [42], which introduces task-based learning for stochastic optimization, and Elmachtoub and Grigas [46], which proposes the SPO+ loss for structured prediction.

More broadly, this approach is closely related to prescriptive analytics [22], where predictive models are explicitly designed to support decision-making. Recent advances in differentiable optimization [2, 5] have further enabled end-to-end integration of optimization layers within neural networks.

However, decision-aware learning faces significant challenges in sequential and combinatorial settings. In multi-stage stochastic programming, decisions must satisfy non-anticipativity constraints:

$$x_t(\xi_{[t]}) = x_t(\xi'_{[t]}) \quad \text{if } \xi_{[t]} = \xi'_{[t]}, \tag{32}$$

which are difficult to enforce within differentiable learning frameworks. Additionally, the presence of integer variables and non-convex feasible regions complicates gradient-based training. These limitations motivate alternative paradigms that more naturally accommodate sequential decision structure.

4.3 Learning-to-Optimize

Learning-to-optimize (L2O) has emerged as a broad framework for integrating machine learning with optimization in ways that are especially relevant to OR/MS. From an OR perspective, the central question is not only how learning can improve prediction, but also how it can enhance decision quality, computational tractability, and implementability across families of optimization problems. In this tutorial, we organize learning-to-optimize methods into three broad paradigms: (1) learning to generate optimization solutions directly, (2) learning to accelerate optimization algorithms, and (3) learning to adapt surrogate optimization models. This organization is broadly consistent with the recent overview of Chen et al. [33].

The first paradigm aims to learn the *solution operator* itself. Given an input instance s, the optimization model returns an optimal decision

$$x^*(s) = \arg\min_{x \in \mathcal{X}(s)} c(x, s), \tag{33}$$

where $\mathcal{X}(s)$ is the feasible set and $c(x, s)$ is the objective function. Rather than solving this problem from scratch for every new instance, one trains a model

$$\hat{x}(s) = f_\theta(s) \tag{34}$$

to map instance features directly to optimal or near-optimal decisions. This setting is particularly attractive in OR/MS applications where closely related problems must be solved repeatedly under changing data, such as online, parametric, and real-time decision environments [16, 137, 138].

As illustrated in Figure 5, the workflow is simple: solved instances $(s, x^*(s))$ are generated offline, these input–solution pairs are used to train a predictive model, and the trained model is then applied to a new instance s to produce a high-quality decision $\hat{x}(s)$. From an OR perspective, this can be viewed as amortizing optimization effort across a family of related instances.

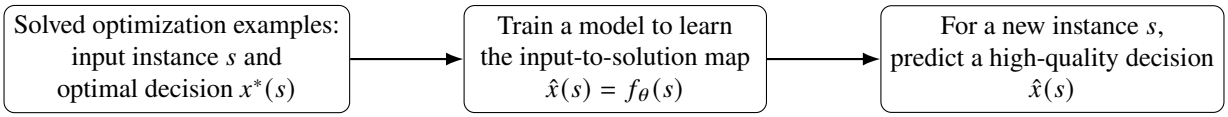


Figure 5: High-level learning-to-optimize pipeline for direct solution generation. Solved optimization instances are used to train a model that maps a problem input directly to an optimal or near-optimal decision, enabling fast prediction for new instances.

The second paradigm uses learning to accelerate, rather than replace, optimization. Here the solver remains central, but selected components are learned from data to improve performance. Examples include learning branching rules, node selection strategies, cutting-plane policies, primal heuristics, and warm starts in combinatorial optimization [16, 70, 71, 87]. This paradigm is especially appealing in OR/MS because it preserves the structure of classical algorithms while improving their practical scalability.

The third paradigm, which we call *Predict-and-Optimize*, adapts the optimization model itself to incorporate predictions. In many real applications, the nominal optimization formulation is only an approximation of the underlying decision environment. Learning or predictive modeling can therefore be used to construct surrogate objectives, approximate value or recourse functions, or embed predictive models directly within the optimization problem:

$$\min_{x \in \mathcal{X}} \hat{c}_\theta(x) \qquad \text{or} \qquad y = f_\theta(x),\ x \in \mathcal{X}. \tag{35}$$

For OR/MS, this paradigm is especially important when data can make optimization models more realistic, adaptive, and decision-relevant [6, 22, 50, 93]. Recent work on two-stage stochastic programming follows this same logic by approximating the expected second-stage value function with a neural network and embedding the resulting surrogate within a tractable optimization model [100]. Related work on optimization over trained neural networks further strengthens this perspective by studying how learned predictive models can be embedded and optimized directly within mathematical programming formulations [118]. More recently, optimization-based formulations of modern learning architectures, including attention mechanisms of transformers represented within mixed-integer nonlinear programming models, further extend this paradigm by bringing predictive structure directly into the optimization layer [84].

Early application examples of joint prediction and optimization include epidemic–logistics optimization for Ebola, where epidemic dynamics are incorporated directly into the decision model [30], and invasive-species applications, where biological growth and spread dynamics are modeled explicitly within optimization-based control and stochastic surveillance frameworks [31, 74].

These paradigms are analytically distinct but often overlap in practice. A direct decision predictor may be followed by a feasibility-repair step, learned policies may be embedded within exact algorithms, and surrogate models may appear inside larger optimization pipelines. In this sense, hybrid ML–optimization methods are best viewed as a cross-cutting design principle rather than a separate category.

This tutorial focuses primarily on the first paradigm: learning direct mappings from problem instances to optimal decisions using deep learning architectures. The emphasis is on structured sequential settings, where decisions must remain feasible, implementable, and consistent with the information available over time under uncertainty. This perspective is distinct from deep reinforcement learning, which is discussed later as a policy-learning approach. The next section develops this direct decision-learning perspective through recurrent, expandable, and non-anticipative learning-optimization architectures [136–138].

5 Learning Optimal Decisions under Constraints

5.1 Sequential Decision Learning via LSTM Frameworks

A natural first instantiation of the direct decision-learning paradigm arises when the optimal solution is itself a sequence. In such settings, the task is not to predict a single static quantity, but to learn a mapping from time-indexed problem data to a time-indexed sequence of decisions. This perspective is especially relevant in OR/MS, where many important optimization problems are inherently sequential and decisions are linked across time through inventories, capacities, budgets, setups, and uncertainty realizations.

LSTM architectures are well suited to this sequence-to-sequence setting because their recurrent memory mechanism updates hidden states over time, allowing the prediction at stage t to depend not only on the current input, but also on information propagated through the sequence [61]. Rather than treating stagewise decisions independently, the network learns how decisions evolve jointly over the planning horizon. Recent work has explored such LSTM-based learning–optimization mappings for sequential decision problems [137].

This is important for multi-stage optimization because dependence arises at two levels. First, the decision at stage t depends on the stagewise data observed over time, such as demands, costs, capacities, or realized uncertainty. Second, and equally importantly, decisions depend on one another across the horizon. In problems such as capacitated lot-sizing, a setup or production decision in one period affects future inventory levels, capacity availability, and subsequent production choices. Through its recurrent state, an LSTM can learn these temporal relationships among decisions, so that the predicted sequence $\hat{x}_1, \ldots, \hat{x}_T$ reflects both the evolution of the input data and the internal logic of the decision process which is impacted by relations of the input data, constraints and the objective function of the optimization formulation.

As illustrated in Figure 6, the bidirectional LSTM architecture maps time-indexed stage information to a structured sequence of stage decisions rather than to isolated pointwise predictions [137]. At stage t, the input vector

$$z_t = \left(\xi_{[t]}, \theta_t, \tilde{x}_{t-1}\right),$$

summarizes the revealed history $\xi_{[t]}$, the stage-dependent model data $\theta_t := (c_t, A_t, H_t, b_t)$, and the previous decision estimate $\tilde{x}_{t-1}$. The sequence is processed by forward and backward recurrent layers, whose hidden representations are concatenated and then mapped to the stage-wise decision prediction $\hat{x}_t(\xi_{[t]})$.

From an OR/MS perspective, the model is not merely fitting correlations in sequential data. Rather, it learns recurring patterns induced by the underlying optimization structure, including temporal coupling

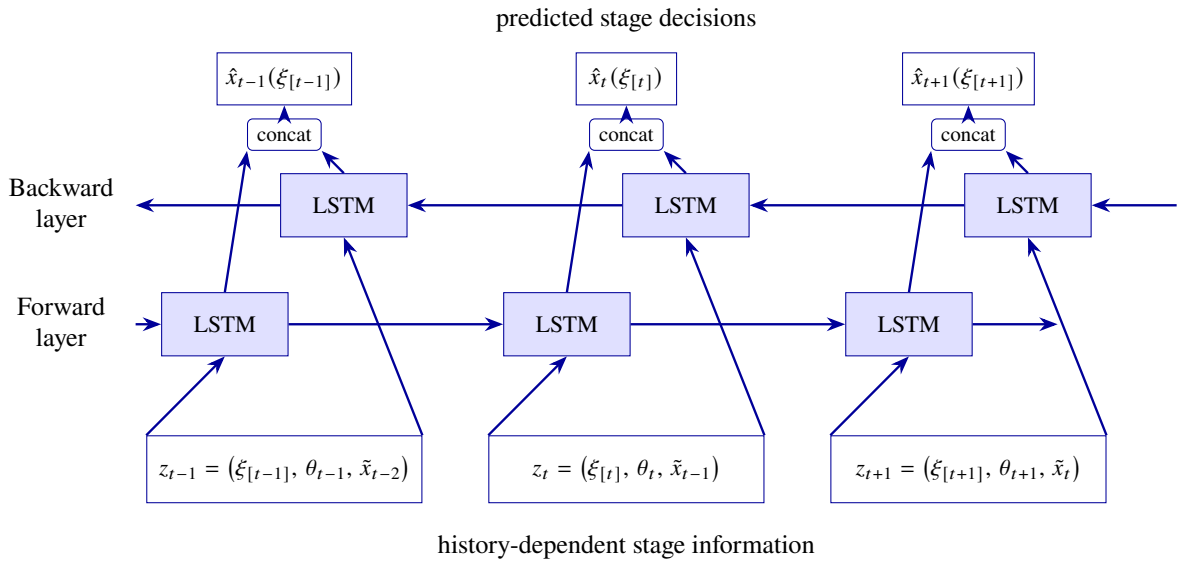


Figure 6: Bidirectional LSTM architecture for sequential decision learning in multi-stage optimization.

across stages, dependencies among decisions, and regularities shaped by balance equations, resource limits, and the objective function itself [16]. Although feasibility is not enforced explicitly within the network, these structural relationships are reflected implicitly in the learned representation. The output is therefore best viewed as a *structured decision trajectory*, in which each stage decision is informed by both the evolving input data and its role in the broader optimization sequence.

Evaluation of learning-to-decision frameworks typically requires a broader set of metrics than classical optimization alone, including not only objective quality, but also feasibility, infeasibility rate, computational gain, the quality of early incumbents, and robustness under downstream decision-making [137, 138].

These properties make LSTM-based learning–optimization frameworks attractive for multi-stage applications in which optimal solutions exhibit strong temporal structure, including production planning, inventory control, energy scheduling, and epidemic response. In such settings, the learned decision sequence can serve either as a fast approximate decision rule or as guidance for downstream optimization through fixing, partial fixing, or warm-starting. This hybrid use is especially appealing in OR/MS, where many important stagewise decisions are binary or mixed-integer in nature—for example, whether to produce, allocate, open, schedule, intervene, or wait. The LSTM captures the temporal structure in these decisions, while the optimization model preserves feasibility, logical consistency, and solution quality.

Although the discussion here centers on LSTM-based frameworks, related work on the capacitated lot-sizing problem suggests that the same learn-to-decision logic can also be pursued with temporal CNNs and transformer-based architectures for dynamic mixed-integer optimization [35, 38].

5.2 Expandable Learning–Optimization Architectures

A central limitation of many learning-based optimization methods is that they are tied to the instance sizes seen during training. For OR/MS applications, however, practical value requires more than high predictive accuracy on a fixed benchmark; it requires learned models that can transfer across longer planning horizons

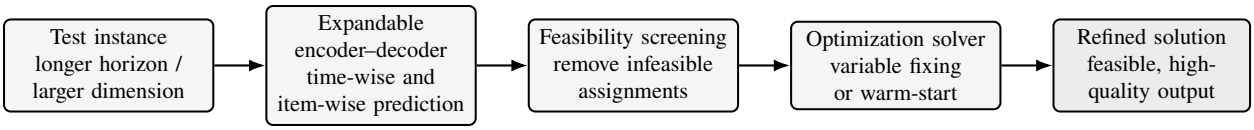


Figure 7: Compact view of the expandable PredOpt architecture. The learned predictor generates partial decisions for a test instance, infeasible assignments are screened out, and the remaining information is used to guide the optimization solver through variable fixing or warm-starting.

and larger problem dimensions without rebuilding the full pipeline. PredOpt [138] addresses this challenge through an expandable learning–optimization architecture that combines a sequence-to-sequence predictor with time-wise and item-wise expansion mechanisms.

As shown in Figure 7, PredOpt follows a modular predict–screen–optimize pipeline. The predictor identifies promising decision patterns, infeasible assignments are screened out, and the resulting partial solution is passed to the optimization solver through fixing or warm-starting. Built on encoder–decoder and attention mechanisms from neural sequence modeling [9, 89, 116], the architecture captures temporal dependencies across stages while also learning recurring structural patterns induced by stage data, coupled decisions, constraints, and the objective. From an OR/MS perspective, this is the key advantage of deep learning in this setting: it does not replace mathematical optimization, but supplements it by learning reusable decision structure that can guide the solver toward high-quality regions of the feasible space.

PredOpt is therefore best viewed as a hybrid decision pipeline rather than a stand-alone predictor. The learned model provides fast guidance, while feasibility screening and solver-based refinement preserve the rigor, feasibility, and interpretability of the underlying optimization model. This complementarity is especially important in sequential decision problems, where scalability matters but implementability cannot be sacrificed.

5.2.1 Generalization Across Horizons and Incremental Expansion

The first source of expandability in PredOpt is generalization across planning horizons. Because the encoder–decoder with attention is not tied to a fixed output length, a model trained on shorter horizons can be applied to longer ones while preserving temporal dependence across stages [9, 89, 138]. This is particularly relevant in OR/MS settings such as rolling-horizon planning, seasonal operations, and adaptive scheduling, where horizon length often changes over time.

Importantly, this expansion is incremental rather than monolithic. The framework does not require end-to-end retraining whenever the horizon changes; instead, it reuses the same trained predictor on longer sequences. In this way, the learned architecture transfers temporal decision structure beyond the training scale, making it more useful for families of related sequential problems rather than a single fixed-size benchmark.

5.2.2 Item-Wise Expansion and Higher-Dimensional Generalization

The second source of expandability is generalization across problem dimension. In many OR/MS problems, the mathematical structure remains the same while the number of items, products, or decision components

changes. PredOpt addresses this challenge through an item-wise expansion mechanism that allows a model trained on smaller instances to be reused on larger ones without retraining [138]. As in Figure 7, the predictor remains embedded in the same predict–screen–optimize pipeline, but its scope is extended across a larger item set through repeated subset-based prediction.

The main idea is to treat the trained predictor as a reusable local decision module rather than as a fixed-size network tied to a single benchmark dimension. Suppose the model is trained on instances with d^M items, while the test instance contains a larger item set $\mathcal{D}$. Instead of predicting all items at once, PredOpt repeatedly samples subsets $S \subseteq \mathcal{D}$ with $|S| = d^M$, constructs the restricted input for each subset, and applies the trained model to obtain local predictions. To preserve the structure of the original problem, the associated constraints are adjusted so that the restricted instance reflects the resource tightness of the full model. For example, in MCLSP the period capacity is scaled by the subset's share of total demand, whereas in MSMK each knapsack capacity is scaled by the subset's share of aggregate item–knapsack weight [138].

Each item appears in multiple overlapping subsets and is therefore predicted under multiple local contexts. These repeated predictions are then aggregated and passed to the downstream optimizer, which restores global feasibility and refines the final solution. In this way, the deep learning architecture captures local structural regularities in decisions and inputs, while optimization restores system-wide consistency. More broadly, the same idea extends beyond item dimension: whenever a problem admits a subset-based or decomposable representation, similar expansion mechanisms may be possible across other structural dimensions, such as locations, customer groups, or commodity classes.

Algorithm 1 Subset-Based Item-Wise Expansion for Larger-Dimensional Instances [138]

Input: trained predictor M built on item dimension d^M; test-instance data α; full item set $\mathcal{D}$; minimum coverage threshold δ; aggregation rule $\mathcal{A}(\cdot)$.
Output: aggregated prediction $\hat{x}_j$ for each item $j \in \mathcal{D}$.

1. Initialize cumulative predictions $\bar{x}_j \leftarrow 0$ and prediction counts $\gamma_j \leftarrow 0$ for all $j \in \mathcal{D}$.
2. **while** there exists an item $j \in \mathcal{D}$ with $\gamma_j \leq \delta$ **do**
   (a) Sample a subset $S \subseteq \mathcal{D}$ such that $|S| = d^M$.
   (b) Construct the restricted input α_S for subset S.
   (c) Make a forward pass with the trained predictor:
   $$\hat{x}_S = M(\alpha_S).$$
   (d) For each $j \in S$, update
   $$\bar{x}_j \leftarrow \bar{x}_j + \hat{x}_j, \qquad \gamma_j \leftarrow \gamma_j + 1.$$
3. For each $j \in \mathcal{D}$, compute the final aggregated prediction
   $$\hat{x}_j \leftarrow \mathcal{A}\big(\bar{x}_j, \gamma_j\big).$$
4. Optionally pass the aggregated prediction $\hat{x}$ to a downstream optimization model for feasibility restoration and solution refinement.

Algorithm 1 summarizes the item-wise expansion mechanism. Rather than forcing the learned model to operate directly on an unseen larger dimension, PredOpt repeatedly applies it to overlapping subsets

whose size matches the training dimension and then aggregates the resulting local predictions into a larger-dimensional partial solution. The learned model transfers local decision structure across dimensions, while optimization recovers feasibility and improves solution quality at the global level.

5.2.3 Computational Evidence and Practical Scalability

The computational results in [138] show that expandability translates into substantial practical gains. For MCLSP with 12 items, PredOpt achieves about 99% prediction accuracy and only 0.04%–0.11% optimality gap across horizons from $T = 30$ to $T = 150$, while requiring only 4.6 seconds at $T = 150$ compared with 25,085.3 seconds for CPLEX. In MSMK, PredOpt remains competitive as dimension grows, with roughly 91.6%–93.8% prediction accuracy, small optimality gaps, and solution times of only seconds on instances for which direct CPLEX runtimes become orders of magnitude larger. These results suggest that expandability is not merely a modeling convenience; it provides meaningful computational leverage on longer and larger sequential decision problems.

In sum, expandable learning–optimization architectures address a central OR/MS challenge: how to transfer learned decision structure beyond the training scale while preserving feasibility and solution quality. PredOpt illustrates how deep learning architectures can capture recurring temporal and structural patterns in sequential decisions, while optimization enforces global consistency and refinement. The result is not just a predictor, but a scalable decision pipeline that remains anchored in the logic of mathematical optimization [138].

5.3 Non-Anticipative Learning–Optimization Frameworks

Sequential decision-making under uncertainty requires decisions to remain implementable as information is revealed over time. In multi-stage stochastic programming, this is captured by *non-anticipativity*: if two scenarios share the same realization history up to stage t, then the stage-t decision must also be the same [113]. Standard sequence models do not enforce this requirement and may therefore exploit distinctions across scenarios that are not yet observable, producing predictions that are statistically plausible but operationally invalid. To address this challenge, Yilmaz and Büyüktahtakın [136] propose the *Non-anticipative Encoder–Decoder with Attention* (NEDA), which adapts the attention-based encoder–decoder architecture of Luong et al. [89] to the scenario-tree structure of multi-stage stochastic programs.

The NEDA framework is intuitive: at stage t, scenarios that are still indistinguishable to the decision maker should also be indistinguishable to the neural network. Let $h^e_{t,s} = [\overrightarrow{h}^e_{t,s}, \overleftarrow{h}^e_{t,s}]$ denote the encoder hidden state for scenario s at stage t, and let $\Omega_{t,s}$ be the set of scenarios that share the same information history up to stage t as scenario s. NEDA enforces non-anticipativity by replacing the scenario-specific encoder state with a scenario-group average:

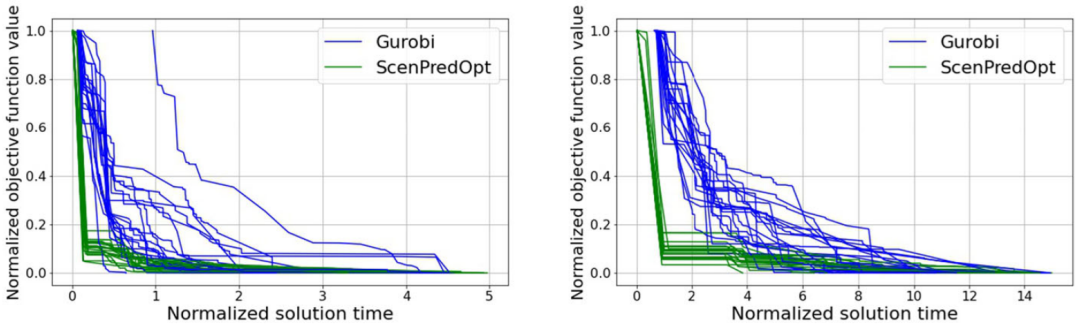
$$\bar{h}^e_{t,s} = \frac{\sum_{s' \in \Omega_{t,s}} h^e_{t,s'}}{|\Omega_{t,s}|}. \tag{36}$$

Because all scenarios in the same information set use the same averaged hidden state, the decoder receives

identical information for indistinguishable scenarios and therefore produces the same stage-t prediction by construction. In effect, the model is prevented from encoding differences that the decision maker has not yet observed.

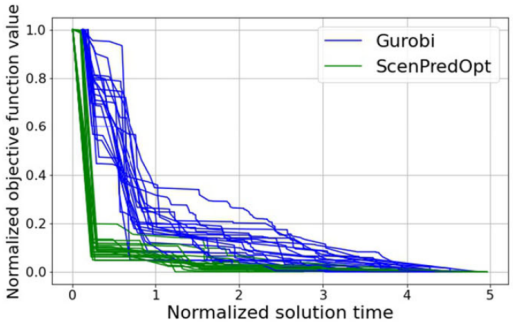
This is the key innovation over deterministic frameworks such as PredOpt. NEDA does not merely learn temporal structure across stages; it also aligns hidden-state representations across the scenario tree so that the learned policy respects the information structure of the stochastic program. When the averaged hidden states are highly consistent, the model can make confident common predictions; when they are not, the predictions become less decisive and the downstream ScenPredOpt framework relies more heavily on feasibility repair, LP-based heuristics, and solver refinement to recover an implementable high-quality solution [136].

Computational Performance. Figure 8 shows that ScenPredOpt reduces the normalized objective value faster and more consistently than Gurobi across all instance classes. The gap widens as uncertainty and problem size increase, with ScenPredOpt reaching near-optimal solutions in a small fraction of the total runtime while Gurobi exhibits slower and more variable convergence. This behavior highlights the practical value of integrating non-anticipativity and feasibility within the learning–optimization pipeline.

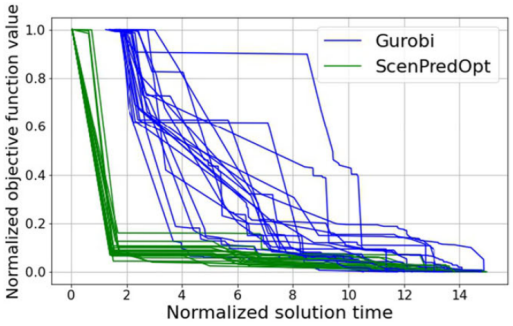


(a) 8 items with 81 scenarios.

(b) 8 items with 243 scenarios.



(c) 12 items with 81 scenarios.



(d) 12 items with 243 scenarios.

Figure 8: Normalized objective-value trajectories for Gurobi and ScenPredOpt across representative instances. ScenPredOpt attains stronger early incumbents and faster practical improvement during the initial stages of the solution process. Adapted from Yilmaz and Büyüktahtakın [136] under the Creative Commons Attribution 4.0 International License (CC BY 4.0).

Across the reported stochastic test sets, ScenPredOpt delivers the strongest practical speed–quality tradeoff relative to direct Gurobi solves and benchmark methods, including Progressive Hedging (PH)[126], SDDiP [4, 141] when applicable, and tailored heuristics [1, 20, 136, 141]. In the SMSMK experiments, it reduces Gurobi runtimes from 1155–7200s to just 1–69s while maintaining small optimality gaps of only 0.77%–1.31% and prediction accuracies above 91% [136]. PH can sometimes achieve smaller gaps [126], but it remains substantially slower, whereas the benchmark heuristics are generally faster than PH but much less accurate. Thus, the main computational advantage of ScenPredOpt is its ability to identify strong incumbents almost immediately while preserving near-optimal solution quality, especially in larger, scenario-rich settings where exact and decomposition-based methods become increasingly expensive.

A similar pattern holds for the 10-item instances. ScenPredOpt reduces runtime from 2351s to 1s for 32 scenarios, from 5130s to 4s for 81 scenarios, from 7200s to 34s for 243 scenarios, and from 7200s to 69s for 512 scenarios, yielding time-improvement factors of 2374, 2047, 329, and 150, respectively. The corresponding optimality gaps remain small at 1.10%, 1.11%, 0.82%, and 0.77%, with prediction accuracy ranging from 91.45% to 95.70% (average 93.73%). In contrast, PH requires 1920s, 1958s, 5670s, and 7406s on these same instances, while the enhanced Bertsimas–Demir heuristic [20] requires 3s, 11s, 66s, and 168s but incurs substantially larger gaps of 2.83%, 2.30%, 2.37%, and 2.11%. On average, ScenPredOpt requires 6s for the 8-item instances and 21s for the 10-item instances, compared with 4016s and 5604s for Gurobi, 2981s and 3739s for PH, and 67s and 48s for the heuristic, while keeping the average optimality gap at only 1.14% and 1.01%, respectively.

A related large-scale direction embeds transformer-based learning within a Benders-type decomposition framework [15] for stochastic combinatorial optimization, illustrating how learned models can accelerate structured exact methods rather than replace them [36].

6 Deep Reinforcement Learning for Sequential Decision Making

Deep reinforcement learning (DRL) is especially appealing in OR/MS settings where uncertainty unfolds over time, explicit stochastic models are difficult to specify, or exact optimization becomes computationally prohibitive. From this perspective, DRL can be viewed as a policy-learning framework that maps states to actions through repeated interaction with an environment, while deep neural networks provide the approximation power needed to represent complex value functions and policies in large, high-dimensional decision spaces. This section focuses on DRL as data-driven policy approximation, its integration with simulation, and its use in stochastic and combinatorial optimization settings. Its effectiveness in OR, however, depends critically on how well the underlying problem structure is encoded in the state, action, reward, and transition dynamics. This is particularly important in simulation-based control, stochastic programming, and combinatorial decision problems, where feasibility, temporal coupling, and recourse structure remain central.

6.1 DRL as Data-Driven Policy Approximation for Stochastic Decision Problems

Building on the MDP, DP, and RL foundations introduced in Section 2, we now focus on DRL as a scalable policy-learning framework for sequential decision problems under uncertainty. Whereas classical DP

characterizes optimality through the Bellman recursion [12], DRL replaces exact value-function computation and full state enumeration with sample-based learning and deep function approximation. This makes it especially relevant for OR/MS settings with large state spaces, sequential uncertainty, and complex dynamics, where exact dynamic programming or stochastic optimization becomes computationally impractical.

In value-based methods, the action-value function $Q^{\pi}(s, a)$ is updated using temporal-difference learning, as in Q-learning [124]. With one-step target

$$y_t = r_t + \gamma \min_{a'} Q_t(s_{t+1}, a'), \tag{37}$$

the update is

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha \big[y_t - Q_t(s_t, a_t) \big]. \tag{38}$$

Deep Q-Networks (DQN) [94] replace tabular value functions with neural approximators $Q(s, a; \theta)$, enabling learning in large state spaces. Policy-based methods instead parameterize the decision rule directly through $\pi_\theta(a \mid s)$ and optimize it with gradient-based methods [130], while actor–critic approaches combine value approximation with policy optimization to improve learning stability [77].

The transition from RL to DRL occurs when deep architectures are used to represent value functions or policies. When the full system state is not directly observable, the natural extension is a partially observable Markov decision process (POMDP), in which decisions are based on observations or belief states rather than fully observed states [66]. In this setting, recurrent deep RL methods such as Deep Recurrent Q-Networks (DRQN) provide a natural bridge by using memory to summarize observation histories and approximate decision-relevant latent state information [60]. For OR/MS applications, this matters because many sequential decision problems involve long-range temporal dependence, partial observability, and high-dimensional state descriptions. Recurrent neural networks, LSTMs [61], and attention-based transformers [121] provide the ability to capture evolving state information, intertemporal dependencies, and longer-horizon effects that are often central to planning, control, and resource allocation problems.

From an OR/MS standpoint, DRL may be interpreted as a simulation-based approximation to stochastic dynamic programming and, more broadly, to multi-stage stochastic programming (MSP). In MSP, decisions are optimized over scenario trees subject to non-anticipativity and recourse structure. DRL replaces this explicit scenario-tree representation with a parameterized policy

$$x_t = \pi_\theta(s_t),$$

where the state s_t summarizes the information available at stage t. In this sense, non-anticipativity is handled implicitly through the state representation, and expectations are approximated through sampling rather than explicit integration over scenarios.

This viewpoint clarifies both the promise and the limitations of DRL for OR/MS. Its strength lies in scalability: it avoids the exponential growth of scenario trees and can learn in rich simulation environments where exact modeling is impractical. Its limitation is that feasibility, integrality, and structural consistency are not guaranteed by default. As a result, effective uses of DRL often combine policy learning with optimization structure, simulation models, or feasibility-enforcing mechanisms, so that the flexibility of

learning is complemented by the rigor of mathematical optimization.

6.2 DRL Integration with Simulation

A particularly important role for deep reinforcement learning in operations research arises when the decision environment can be simulated with high fidelity but cannot be represented through a tractable analytical transition model. In such settings, the main difficulty is not the absence of system knowledge, but the inability to express the dynamics in a form amenable to classical dynamic programming, stochastic programming, or other policy-based optimization methods. Simulation-integrated DRL addresses this challenge by treating a simulation model as the learning environment and training a policy through repeated interaction with that environment [28]. This creates a practical bridge between mechanistic system modeling and sequential decision optimization, especially in settings that are nonlinear, path-dependent, partially observed, or shaped by heterogeneous interacting agents [94, 102, 103, 117].

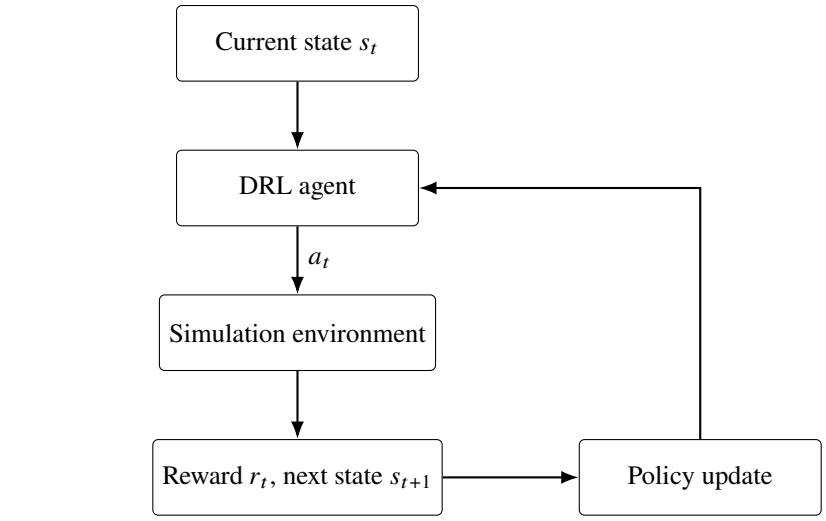
From a decision making perspective, this paradigm expands the scope of sequential optimization beyond problems that can be summarized by compact state transitions or scenario trees. Rather than requiring a closed-form stochastic model, one can rely on a simulation engine that already captures operational rules, feedback effects, and domain-specific complexity. The DRL agent then learns a policy directly from the simulated evolution of the system, with reward signals encoding the underlying objectives and tradeoffs. In this sense, simulation-integrated DRL can be viewed as a policy-learning layer built on top of a computational model of the system [102, 103, 117].

Figure 9 provides two complementary views of this integration and makes clear why simulation becomes part of the policy-learning loop [28]. Panel (a) presents the high-level closed-loop workflow linking the current state, the DRL agent, the simulation environment, the resulting reward and next state, and the subsequent policy update. Panel (b) makes the sequential structure explicit over time. At each decision epoch t, the simulator provides a state description s_t, the DRL agent selects an action a_t, and the simulator propagates the consequences of that action to generate a reward r_t together with the information needed to construct the next state s_{t+1}. Thus, the learning process unfolds through the repeated interaction

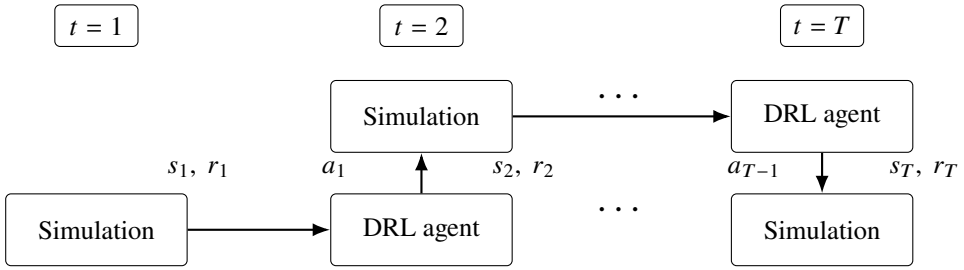
$$s_t \;\rightarrow\; a_t \;\rightarrow\; (r_t, s_{t+1}), \qquad t = 1, \ldots, T-1,$$

over the planning horizon. The notation emphasizes that the simulator replaces an explicit analytical transition kernel by directly mapping actions into future system outcomes [28, 103, 117].

Epidemic control provides a compelling example. In such problems, disease spread depends on rich interactions among contact behavior, intervention timing, vaccination, compliance, and population heterogeneity. These effects are difficult to encode through explicit transition equations, yet they can often be represented naturally through agent-based simulation. Bushaj et al. [28] operationalize this idea through a simulation-deep reinforcement learning framework in which a DRL agent interacts with an agent-based epidemic simulator, so-called Covasim [69] to learn adaptive intervention policies under multi-objective reward tradeoffs. This modeling logic is also consistent with related DRL-based epidemic-control studies that learn mitigation policies in large-scale simulated environments [85].



(a) High-level workflow



(b) Sequential time-indexed view

Figure 9: Two complementary views of simulation-integrated deep reinforcement learning. Panel (a) presents the high-level closed-loop workflow linking the DRL agent, the simulation environment, and policy updates. Panel (b) presents the same framework in time-indexed form, emphasizing the sequential evolution of states, actions, and rewards across decision epochs. Adapted and generalized from Bushaj et al. [28].

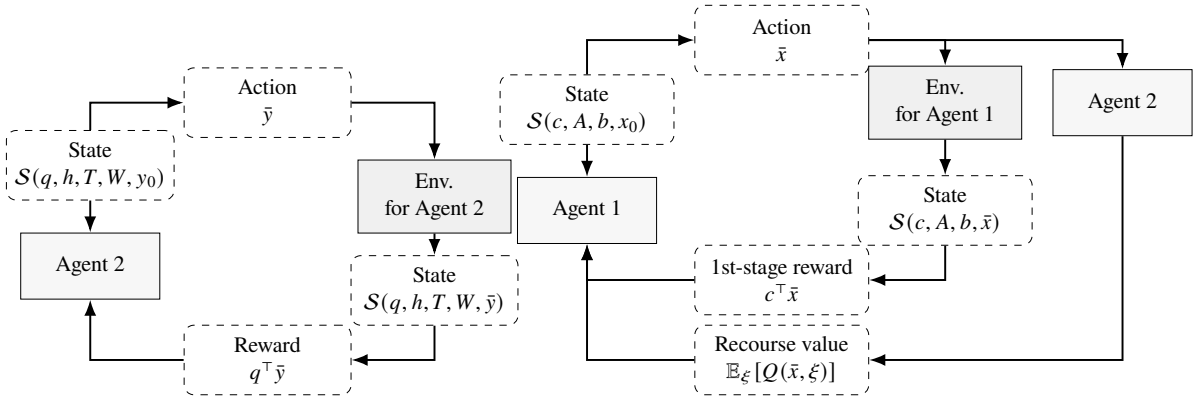
More broadly, simulation-integrated DRL turns complex simulators into decision laboratories. Rather than using simulation only for post hoc evaluation, it uses simulation as part of the optimization loop to learn responsive, adaptive, and operationally meaningful policies in environments where explicit stochastic formulations are either unavailable or too restrictive for practical solution [28, 102, 103]. The resulting framework provides a natural mechanism for learning responsive and adaptive policies under uncertainty, particularly in applications where explicit stochastic models are unavailable or intractable.

6.3 DRL for Stochastic Programs

Another important direction is the use of DRL to approximate solutions to stochastic programs. A leading example is Yilmaz and Büyüktahtakın [135], which recasts a two-stage stochastic program as a multi-agent DRL framework. The central idea is to replace the repeated solution of a large scenario-based optimization model with learned stage-wise decision policies. This is especially appealing in two-stage stochastic programming, where the problem naturally separates into here-and-now decisions and recourse decisions taken after uncertainty is revealed.

The key OR/MS insight is that the second-stage recourse decision can be represented not only as a nested optimization problem, but also as a learnable decision process. In the framework of Yilmaz and Büyüktahtakın [135], Agent 1 learns the first-stage decision x, while Agent 2 learns the second-stage recourse decision y conditional on the first-stage decision and the realized uncertainty. In this way, the logic of stochastic programming is preserved, but repeated optimization is replaced by learned decision rules.

Figure 10 illustrates the training structure. Panel (a) shows the training loop for Agent 2, which learns a recourse policy after a scenario is realized. Starting from a second-stage state, denoted schematically by $\mathcal{S}(q, h, T, W, y_0)$, Agent 2 chooses a recourse action $\bar{y}$, transitions to the updated state $\mathcal{S}(q, h, T, W, \bar{y})$, and receives a reward based on the resulting second-stage value, here represented by $q^\top \bar{y}$. Thus, Agent 2 learns how to map realized second-stage uncertainty into corrective recourse actions. Panel (b) highlights the more distinctive feature of the framework: Agent 1 is trained not only on the immediate first-stage contribution $c^\top \bar{x}$, but also on the downstream value induced by that decision, represented by $\mathbb{E}_\xi[Q(\bar{x}, \xi)]$. Beginning from the first-stage state $\mathcal{S}(c, A, b, x_0)$, Agent 1 selects a first-stage action $\bar{x}$, which is then passed forward to both the first-stage environment and Agent 2. The learning signal therefore reflects both immediate performance and future recourse consequences. In OR/MS terms, Agent 1 learns a here-and-now policy that internalizes expected downstream recourse value.



(a) Agent 2 training. (b) Agent 1 training.

Figure 10: Two-stage DRL training overview. Agent 2 learns recourse actions after scenario realization, while Agent 1 learns first-stage decisions using both immediate reward and downstream recourse feedback. Adapted from Yilmaz and Büyüktahtakın [135].

This interaction between Agent 1 and Agent 2 is what makes the framework especially compelling. Agent 2 learns how to respond once uncertainty is observed, while Agent 1 learns to anticipate that future response when making the initial decision. The result is a policy-based approximation to the recourse mapping: rather than evaluating many scenarios through repeated optimization, the framework learns how first-stage choices shape second-stage performance and improves both policies through interaction over time. In this sense, DRL provides a new computational lens for stochastic programming, replacing explicit scenario enumeration with learned stagewise policies that remain sensitive to inter-stage dependence [102, 103, 117, 135].

The computational motivation is equally important. The two-stage DRL framework can reduce solution

times from hours to fractions of a second while maintaining moderate optimality gaps, roughly 6% to 10% across the reported test sets [135]. These results highlight both the promise and the limitation of DRL for stochastic programs. Learned policies can provide extremely fast decisions when speed is critical, such as dynamic pricing and routing but they do not offer the same optimality guarantees as exact stochastic programming methods. For this reason, DRL is best viewed not as a replacement for stochastic programming, but as a complementary approximation paradigm for large-scale or real-time settings where rapid, reasonably high-quality solutions are essential.

6.4 DRL for Combinatorial Optimization

Deep reinforcement learning has emerged as a promising approximation framework for combinatorial optimization, where decisions are discrete, constraints are tight, and exhaustive search quickly becomes impractical. Early studies showed that neural policies can learn strong solution patterns for structured problems such as routing and sequencing [13, 78, 97]. The main lesson for optimization, however, is that performance depends less on the generic DRL algorithm than on how well the learning environment reflects the structure of the underlying model, including feasibility, constraint interactions, and the organization of the decision space [16, 27, 92].

A particularly relevant example is Bushaj and Büyüktahtakın [27], which develops a structure-aware DRL framework for the multi-dimensional knapsack problem (MKP). The contribution is not simply the use of DRL on a binary integer program, but the redesign of the learning environment so that it encodes the combinatorial structure of the model. The framework combines three elements: a K-means-based procedure for constructing a feasible reduced model, a heuristic variable-reordering step, and a reordered two-dimensional state representation on which several DRL algorithms—Advantage Actor–Critic (A2C), Actor Critic using Kronecker-Factored Trust Region (ACKTR), Deep Q-Network (DQN), and Proximal Policy Optimization (PPO2)—are trained and compared. The broader point is that initialization, state design, and feasibility support can materially improve policy learning when they are embedded directly into the pipeline rather than left to post-processing.

The K-means component is used to construct a reduced but informative relaxation of the MKP. Each constraint $i \in \mathcal{I}$ is represented by the $(n+1)$-dimensional vector

$$d_i = [a_{i1}, \ldots, a_{in}, b_i],$$

where a_{ij} is the coefficient of item j in constraint i and b_i is the corresponding right-hand side. Similarity between two constraints d_f and d_k is measured by the Euclidean distance

$$D^2_{d_f, d_k} = \sum_{j=1}^{n+1} (d_{fj} - d_{kj})^2.$$

The constraints are clustered, representative constraints are selected, and the resulting reduced model is solved by CPLEX. If the solution violates omitted constraints, the most violated ones are added back recursively until feasibility for the original model is recovered. Thus, K-means is used not to produce the final answer,

but to generate a feasible relaxation that provides the DRL environment with a structured and inexpensive starting point.

Figure 11 summarizes the pipeline. Training instances are processed through two coordinated paths: a K-means-based feasible initialization path and a heuristic variable-reordering path. Once a feasible reduced solution is obtained, it is merged with the reordered representation to form a two-dimensional RL environment. Candidate DRL algorithms are then trained on this environment. In effect, the MKP is not presented as an unstructured sequence of independent 0–1 choices; instead, the environment exposes patterns in the constraint system and arranges the decision variables so that the agent can learn how local item selections affect global feasibility and objective value.

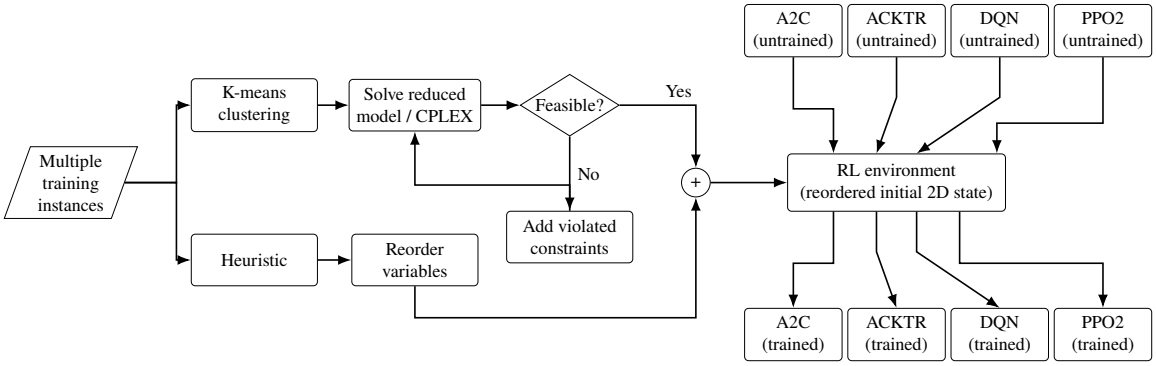


Figure 11: Structure-aware DRL pipeline for the multi-dimensional knapsack problem. Training instances are processed through a K-means-based feasible relaxation path and a heuristic variable-reordering path, then transformed into a reordered two-dimensional RL environment on which candidate DRL algorithms are trained. Adapted and simplified from Bushaj and Büyüktahtakın [27].

An important modeling insight is that the strongest initial heuristic is not always the best learning signal. The authors report that higher-quality classical heuristics can reduce exploration and encourage premature convergence. The K-means initialization therefore serves a more balanced role: it supplies a feasible and informative benchmark without overly constraining the agent's search behavior.

The computational payoff is substantial. Bushaj and Büyüktahtakın [27] show that this design yields major speedups over CPLEX on medium and large MKP instances while maintaining very small optimality gaps and strong agreement with high-quality item-selection patterns. More broadly, the framework illustrates a useful principle for combinatorial optimization: DRL is most effective when it is used as a structure-aware policy layer, supported by informed initialization, feasibility-preserving design, and hybrid interaction with optimization logic. This points to promising directions such as DRL-guided primal heuristics, warm-start generation, feasibility-aware local improvement policies, and tighter integration with exact methods including branch-and-bound, cutting planes, and decomposition [16, 27, 92]. Across simulation-based systems, stochastic programs, and combinatorial optimization, DRL is most useful not as a substitute for optimization, but as a complementary framework for learning high-quality decision policies within structure-aware formulations.

7 Applications and Interdisciplinary Impact

The practical importance of the learning–optimization interface is already evident in application domains where decisions are repeated, information unfolds over time, and operational constraints cannot be ignored. Recent work spans predict-then-optimize, contextual prescriptive analytics, decision-focused learning, neural combinatorial optimization, and reinforcement-learning-based decision systems [16, 45, 46, 67, 90]. Despite their methodological differences, these approaches share a common principle: learning enables the transfer of experience across related instances and adaptation to changing environments, whereas optimization remains indispensable for enforcing feasibility, coordinating decisions, and preserving system-level performance.

7.1 Supply Chains, Service Systems, and Procurement

Supply chains continue to provide one of the clearest testbeds for these ideas because decisions recur over time, propagate across networks, and must be made under uncertainty. In routing and related combinatorial problems, pointer and attention-based architectures have demonstrated that learned policies can generate high-quality solutions across broad classes of instances [13, 78, 123]. In inventory and replenishment settings, deep reinforcement learning has shown considerable promise when classical exploitable structure is difficult to leverage directly, while recent hybrid methods integrate policy learning with integer programming to manage large constrained action spaces [37, 56, 59, 98]. Related advances in queueing, procurement, and sequential production–inventory systems further indicate that learned surrogates and integrated learning–optimization frameworks can support repeated operational decisions without requiring each new instance to be solved from scratch[26, 40, 91, 136–138]. Recent review evidence identifies supply chains as a major domain for AI–optimization integration [52], while related work on supply chain viability and resilience, including intertwined supply networks and AI-enabled intelligent digital twins, further underscores the breadth of this integration [63, 64].

7.2 Healthcare and Epidemic Response

The value of integrating learning with decision structure becomes especially evident in healthcare and the response to epidemics. In clinical decision support, reinforcement learning and related sequential models must contend with partial observability, confounding, and delayed outcomes, so MDP-based formulations provide an important foundation, but not a complete solution [76, 86, 111]. Related work on HIV prevention and treatment, epidemic resource allocation with equity considerations, integrated epidemic–supply chain planning, vaccine distribution, and other healthcare logistics settings further illustrate the breadth of this interface, spanning approximate dynamic programming, multi-stage stochastic optimization, risk-averse planning, and hybrid clustering and reinforcement-learning-based methods [37, 39, 139, 140]. At the population level, epidemic control introduces spatial interdependence, limited resources, and the need for coordinated intervention over time, making learning more effective when it is embedded within simulation and optimization rather than used as a stand-alone policy engine [28]. Early examples of the *Predict-and-Optimize* phenomenon also appeared in healthcare and epidemic applications, including Ebola logistics,

cancer treatment planning under spatio-temporal tumor growth, fairness-driven epidemic allocation, and data-driven COVID-19 ventilator allocation under uncertainty and risk [30, 65, 73, 140].

7.3 Energy, Agriculture, and Environmental Systems

Energy and infrastructure systems benefit similarly from hybrid approaches that couple rapid approximation with physical and operational discipline. Learning-based surrogates and data-driven decision models are increasingly used to accelerate repeated planning and control tasks in power systems and microgrids, including optimal power flow and real-time energy management applications [49, 96, 99]. Agriculture offers an equally instructive example: irrigation, fertilization, treatment, and harvest decisions are inherently dynamic, resource-constrained, and state-dependent, and are now being explored through reinforcement-learning-based decision support [54]. Comparable opportunities arise in environmental management, where decisions must adapt over time while remaining consistent with ecological processes, spatial interactions, and budget limitations.

7.4 Robotics, UAVs, and Other Emerging Applications

Robotics, unmanned aerial vehicles (UAVs), and autonomous delivery systems extend these questions into real-time operational environments. In such settings, learned policies must react quickly while respecting energy, synchronization, routing, and service constraints. Recent work on UAV routing and truck–drone coordination illustrates the promise of reinforcement learning and learning-based heuristics in exactly this regime [48, 131]. The significance of these applications lies not in replacing optimization, but in broadening the reach of sequential decision models to environments where speed, uncertainty, and combinatorial structure interact in more immediate and consequential ways.

Across these domains, a clear message emerges. Learning contributes more than prediction: it extracts structure from data, transfers experience across related problems, and adapts decisions as conditions evolve. Mathematical grounding anchors such as adaptivity in constraints, recourse, coordination, and system-level objectives. Together, these advances show that high-impact decision-making will be shaped not by learning or optimization alone but by their principled integration. They also highlight the inherently interdisciplinary nature of this interface and its growing role in designing intelligent, scalable, and implementable decision systems for complex societal challenges.

8 Open Challenges and Research Frontiers

8.1 Generalization, Scale, and Evaluation

A central challenge in learning-based decision-making is generalization across instances, horizons, and scales. Models may perform well on the distributions on which they are trained, yet degrade on larger or structurally different problems. This is especially pronounced in combinatorial and sequential settings, where decision quality must be preserved as size, uncertainty, and the planning horizon change.

The deeper issue is what it means for a decision model to generalize. Unlike prediction tasks, where the approximation error is measured, decision-making requires the preservation of feasibility, structure, and solution quality under changing conditions. Architectures such as LSTMs and transformers can capture temporal and structural dependence, but do not ensure consistent performance across scales. Promising directions include expandable architectures, representations that respect permutation invariance and structural symmetries in decision problems, and training schemes that align learning more directly with downstream decision quality [16, 42, 121, 129, 136–138].

Progress also depends on a more rigorous evaluation. Learning–optimization methods are still frequently tested on limited or highly synthetic datasets, which makes meaningful comparison difficult. More informative benchmarks should assess not only solution quality, but also feasibility, computational efficiency, generalization, calibration, and robustness to distribution shift across a wider range of realistic application domains [16, 42, 129].

8.2 Feasibility, Reliability, and Trust

Feasibility and reliability remain central in the deployment of learned decision models, especially in high-stakes settings. Unlike optimization models, which explicitly enforce constraints, learning-based systems may produce infeasible decisions unless constraints are carefully incorporated into the architecture or training process.

This is particularly challenging when decisions involve discreteness, combinatorial structure, or non-anticipativity. Penalty methods, projection layers, and differentiable surrogates are useful, but they may introduce an approximation error or an additional computational burden. Therefore, a natural direction is to combine the learned proposals with optimization-based repair, certification, or refinement, so that feasibility is handled structurally rather than heuristically [2, 5, 50, 136]. More broadly, stronger guarantees are still needed on feasibility, robustness, and suboptimality under uncertainty.

Trust is equally important. Decision-makers must understand not only what is recommended but also why it is reasonable and under what conditions it may fail. Classical optimization models often offer a degree of transparency through explicit objectives, constraints, and sensitivity information. Deep models, on the contrary, may function as opaque approximators. Bridging this gap will require a combination of post-hoc explanation tools and more traditional sensitivity, scenario, and policy analysis [16, 88, 105]. In sequential settings, trust also depends on how learned decisions behave under perturbations, tighter constraints, and alternative uncertainty realizations. Human–AI collaboration is also likely to matter for trust in practice, as evidence from field settings and large-scale meta-analysis suggests that human involvement can improve reliance and decision quality, but only when task design and division of labor align with the complementary strengths of humans and AI [119, 134].

8.3 Integrated Learning, Optimization, and Hybrid Decision Frameworks

A recurring theme throughout this tutorial is that learning and optimization are often most effective when treated as interacting components rather than separate stages. The main question is not whether one should

replace the other, but how the two should interact.

In predict-then-optimize settings, learning supplies inputs to an optimization model; in decision-aware learning, the optimization problem shapes the training objective. More tightly integrated approaches embed optimization layers within neural architectures or use learning to guide branching, decomposition, and search [2, 5, 16, 42, 53, 71, 72, 87, 129]. The trade-off is clear: end-to-end training can become computationally demanding, while loosely coupled pipelines may miss important feedback between prediction and decision quality. A useful path forward is a modular design in which learning provides fast approximations or structural guidance, while optimization preserves feasibility and improves the quality of the final solution [16, 136, 138].

The same logic extends to deep reinforcement learning. Dynamic programming, approximate dynamic programming, and multi-stage stochastic programming derive decisions from explicit models of system dynamics, constraints, and uncertainty [12, 101, 113]. By contrast, DRL learns policies through interaction and is particularly appealing when system dynamics are complex, simulation-based, or only partially known [77, 94, 117, 130]. Its flexibility and scalability are attractive, but DRL does not naturally enforce feasibility, integrality, or non-anticipativity. This limitation has motivated hybrid frameworks in which learning supplies candidate actions, value approximations, or policy guidance, while optimization maintains structural consistency and enhances reliability [16, 50, 71, 87, 136].

8.4 Uncertainty, Robustness, and Risk-Aware Decision-Making

In sequential decision-making, point predictions are rarely enough. The confidence in those predictions must also be quantified and carried into the decision model, particularly when learned components are used to estimate demands, transition dynamics, scenario likelihoods, or recourse behavior.

Several directions are especially relevant. Bayesian approximations and dropout-based methods offer one route to model uncertainty, deep ensembles often provide practical and reasonably calibrated uncertainty estimates, and conformal prediction yields distribution-free predictive sets under appropriate assumptions [7, 51, 68, 80]. These estimates become more useful when translated into scenarios, uncertainty sets, or ambiguity sets within stochastic, robust, and distributionally robust optimization models [14, 21, 41, 95, 128]. In the sequential setting, related formulations have also been developed for distributionally robust Markov decision processes, which introduce ambiguity directly into dynamic decision models [133].

This same line of thought has become increasingly influential in CS and AI, where distributional robustness is now closely tied to adversarial training, worst-group generalization, and performance under distribution shift [44, 110, 115]. More recent work in the optimization literature further connects these ideas to heterogeneous subpopulations and latent covariate shifts [43]. A natural next step is therefore to connect uncertainty-aware learning more directly with risk-sensitive and distributionally robust decision-making, so that learning informs not only expected outcomes, but also the reliability of decisions under model error, rare events, and distribution shift.

8.5 Interdisciplinary and Emerging AI Frontiers

Some of the most promising opportunities now lie within the discipline. Many important decision problems combine sequential uncertainty, large and heterogeneous data streams, human behavior, physical and institutional constraints, and multiscale dynamics in ways that no single methodology can adequately capture. Examples include resilient supply networks, adaptive healthcare and epidemic planning, precision agriculture and food systems, ecological and environmental management, resilience to energy-grid, autonomous logistics, and decision support in biological and biomedical systems. These interfaces also extend naturally to strategic and public-policy settings: recent work on principal–agent games, for example, integrates optimization, machine learning, and game theory to develop interpretable heuristics that support fast and explainable policy decision-making in forest health management [11].

A closely related frontier concerns how recent AI models can expand the way optimization is formulated, explored, and ultimately used. Beyond direct learning of decision policies, large language models and related generative approaches are beginning to support natural language optimization modeling, heuristic generation, algorithmic discovery for hard optimization problems, interactive planning, and model diagnosis or repair [3, 8, 32, 107, 114, 122]. This points to a broader role for AI: not merely as a predictive engine, but as an interface among data, models, algorithms, and decision-makers. At the same time, progress at the intersection of game theory and deep learning shows how deep reinforcement learning and empirical game-theoretic analysis can be combined to learn robust policies in sequential multi-agent environments [81]. Such developments are especially relevant when strategic interaction, adaptation, and learning unfold together over time. Recent work on incorporating safety constraints into multi-agent DRL for naval search and defense applications further illustrates how current DRL approaches can be extended to enforce feasibility explicitly through hard constraints [34].

Another emerging interface links learning and optimization with quantum computing. Hybrid quantum–classical methods, especially variational approaches such as the Quantum Approximate Optimization Algorithm (QAOA), address hard combinatorial optimization problems through parameterized quantum circuits whose parameters are iteratively tuned by a classical optimizer. In this sense, they sit naturally at the boundary of learning and optimization, although the questions of scalability, trainability, and practical advantage remain open [24]. The greatest promise lies in hybrid decision platforms that integrate deep learning, DRL, optimization, generative AI, and, over time, quantum-enhanced methods within a common framework for sequential decision-making under uncertainty.

9 Conclusion

This tutorial has presented an OR/MS-centered view of deep learning for sequential decision-making under uncertainty. Its central message is straightforward: in these settings, prediction alone is not enough. What matters is decision quality under constraints, recourse, uncertainty, and evolving information. This distinction remains fundamental in OR/MS and is equally important in contemporary learning-based systems [16, 42, 46, 129, 138].

From that perspective, the tutorial has brought together several strands of work that are often stud-

ied separately. At the foundational level, dynamic programming, reinforcement learning, and multi-stage stochastic programming provide complementary ways to model sequential decisions under uncertainty [12, 23, 113, 117]. At the architectural level, modern neural models—including feedforward networks, recurrent models, transformers, and deep reinforcement learning—expand the range of mappings, policies, and value functions that can be learned from rich data [28, 57, 117, 121, 135]. At the methodological level, decision-aware learning, learning-to-optimize, expandable architectures, non-anticipative learning, and hybrid DRL–optimization approaches suggest a broader design principle: learning and optimization are often most useful when treated as interacting components of a common decision pipeline rather than as isolated stages [42, 129, 136–138].

A consistent theme throughout has been complementarity. Deep learning contributes to flexible approximation, adaptability, and scalability. Optimization contributes the mathematical structure needed to model constraints, recourse, and uncertainty, together with decision criteria tied to system-level objectives. This complementarity is especially important in large-scale settings, where classical methods remain principled but computationally demanding, and where purely learned policies may be flexible but not yet reliable enough for direct deployment [12, 23, 94, 113]. Viewed in this way, the role of learning is often not to replace optimization, but to strengthen it: by learning useful structure, accelerating repeated solution processes, improving time-to-good decisions, and supporting decision rules that remain compatible with solver-based refinement [16, 71, 87, 136–138].

The applications reviewed here reinforce the breadth of this opportunity. Across supply chains, healthcare and epidemic response, agriculture and environmental systems, energy, robotics, and autonomous mobility, decisions must be made sequentially, under uncertainty, and under operational constraints. These are precisely the settings in which structured decision models and modern learning architectures can be combined most productively. The opportunity is not simply to predict better, but to build adaptive decision systems that learn from data while remaining feasible, uncertainty-aware, and operationally meaningful.

At the same time, the research frontier remains substantial. Important directions include stronger generalization across instances and scales, tighter integration of uncertainty quantification with downstream decision-making, more reliable guarantees for feasibility and robustness, and benchmark environments that better reflect real sequential settings [7, 16, 21, 136]. Emerging interfaces that involve large language models, generative AI, algorithmic discovery, and related tools may also expand the way optimization models are formulated, explored, and communicated [3, 25, 32].

More broadly, the field appears to be moving from predictive AI toward decision-capable AI. In that transition, OR/MS has much to contribute: not only optimization methods but also a rigorous analytical perspective on what high-quality decision-making requires under uncertainty. From this perspective, deep learning for sequential decision-making under uncertainty is best understood not as a departure from OR/MS, but as a natural extension of its core mission: to design decision systems that are analytically grounded, computationally effective and operationally useful.

References

[1] Absi N, van den Heuvel W (2019) Worst-case analysis of relax and fix heuristics for lot-sizing problems. *European Journal of Operational Research* 279(2):449–458, URL http://dx.doi.org/10.1016/j.ejor.2019.06.010.

[2] Agrawal A, Amos B, Barratt S, Boyd S, Diamond S, Kolter JZ (2019) Differentiable convex optimization layers. *Advances in Neural Information Processing Systems*, volume 32, URL http://dx.doi.org/10.48550/arXiv.1910.12430.

[3] Ahmaditeshnizi A, Gao W, Udell M (2024) Optimus: Scalable optimization modeling with (MI)LP solvers and large language models. *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, 577–596 (PMLR), URL https://proceedings.mlr.press/v235/ahmaditeshnizi24a.html.

[4] Ahmed S, Ding L, Shapiro A (2019) A python package for multi-stage stochastic programming. URL https://optimization-online.org/2019/05/7199/, optimization Online, pp. 1–41.

[5] Amos B, Kolter JZ (2017) Optnet: Differentiable optimization as a layer in neural networks. *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 136–145 (PMLR), URL http://dx.doi.org/10.48550/arXiv.1703.00443.

[6] Anderson R, Huchette J, Ma W, Tjandraatmadja C, Vielma JP (2020) Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming* 183(1):3–39, URL http://dx.doi.org/10.1007/s10107-020-01474-5.

[7] Angelopoulos AN, Bates S (2023) Conformal prediction: A gentle introduction. *Foundations and Trends in Machine Learning* 16(4):494–591, URL http://dx.doi.org/10.1561/2200000101.

[8] Ao R, Simchi-Levi D, Wang X (2026) Optirepair: Closed-loop diagnosis and repair of supply chain optimization models with LLM agents. *arXiv preprint arXiv:2602.19439* URL http://dx.doi.org/10.48550/arXiv.2602.19439.

[9] Bahdanau D, Cho K, Bengio Y (2014) Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations*, URL http://dx.doi.org/10.48550/arXiv.1409.0473.

[10] Bai S, Kolter JZ, Koltun V (2018) An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271* URL http://dx.doi.org/10.48550/arXiv.1803.01271.

[11] Baswapuram AK, Chen C, Cai W, Büyüktahtakın İE (2026) An interpretable ensemble heuristic for principal-agent games with machine learning. Working paper.

[12] Bellman R (1957) *Dynamic Programming* (Princeton, NJ: Princeton University Press), ISBN 9780691079516, URL https://press.princeton.edu/books/hardcover/9780691079516/dynamic-programming.

[13] Bello I, Pham H, Le QV, Norouzi M, Bengio S (2017) Neural combinatorial optimization with reinforcement learning. *International Conference on Learning Representations*, URL http://dx.doi.org/10.48550/arXiv.1611.09940.

[14] Ben-Tal A, Nemirovski A (1998) Robust convex optimization. *Mathematics of Operations Research* 23(4):769–805, URL http://dx.doi.org/10.1287/moor.23.4.769.

[15] Benders JF (1962) Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* 4(1):238–252, URL http://dx.doi.org/10.1007/BF01386316.

[16] Bengio Y, Lodi A, Prouvost A (2021) Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research* 290(2):405–421, URL http://dx.doi.org/10.1016/j.ejor.2020.07.063.

[17] Bengio Y, Simard P, Frasconi P (1994) Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* 5(2):157–166, URL http://dx.doi.org/10.1109/72.279181.

[18] Bertsekas DP (1995) *Dynamic Programming and Optimal Control* (Belmont, MA: Athena Scientific), ISBN 9781886529434, URL https://www.athenasc.com/dpcontents.html.

[19] Bertsekas DP, Tsitsiklis JN (1996) *Neuro-Dynamic Programming* (Belmont, MA: Athena Scientific), ISBN 9781886529106, URL https://www.athenasc.com/ndpbook.html.

[20] Bertsimas D, Demir R (2002) An approximate dynamic programming approach to multidimensional knapsack problems. *Management Science* 48(4):550–565, URL http://dx.doi.org/10.1287/mnsc.48.4.550.208.

[21] Bertsimas D, Gupta V, Kallus N (2018) Data-driven robust optimization. *Mathematical Programming* 167(2):235–292, URL http://dx.doi.org/10.1007/s10107-017-1125-8.

[22] Bertsimas D, Kallus N (2020) From predictive to prescriptive analytics. *Management Science* 66(3):1025–1044, URL http://dx.doi.org/10.1287/mnsc.2018.3253.

[23] Birge JR, Louveaux F (2011) *Introduction to Stochastic Programming* (New York, NY: Springer), 2 edition, URL http://dx.doi.org/10.1007/978-1-4614-0237-4.

[24] Blekos K, Brand D, Ceschini A, Chou CH, Li RH, Pandya K, Summer A (2024) A review on quantum approximate optimization algorithm and its variants. *Physics Reports* 1068:1–66, URL http://dx.doi.org/10.1016/j.physrep.2024.03.002.

[25] Brown TB, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A, Agarwal S, Herbert-Voss A, Krueger G, Henighan T, Child R, Ramesh A, Ziegler DM, Wu J, Winter C, Hesse C, Chen M, Sigler E, Litwin M, Gray S, Chess B, Clark J, Berner C, McCandlish S, Radford A, Sutskever I, Amodei D (2020) Language models are few-shot learners. *Advances in Neural Information Processing Systems*, volume 33, 1877–1901, URL http://dx.doi.org/10.48550/arXiv.2005.14165.

[26] Busch N, Crönert T, Minner S, Rettinger M, Sel B (2023) Deep learning for commodity procurement: Nonlinear data-driven optimization of hedging decisions. *INFORMS Journal on Optimization* 5(3):273–294, URL http://dx.doi.org/10.1287/ijoo.2022.0086.

[27] Bushaj S, Büyüktahtakın İE (2024) A k-means supported reinforcement learning framework to multi-dimensional knapsack. *Journal of Global Optimization* 89(3):655–685, URL http://dx.doi.org/10.1007/s10898-024-01364-6.

[28] Bushaj S, Yin X, Beqiri A, Andrews D, Büyüktahtakın İE (2023) A simulation-deep reinforcement learning (SiRL) approach for epidemic control optimization. *Annals of Operations Research* 328(1):245–277, URL http://dx.doi.org/10.1007/s10479-022-04926-7.

[29] Büyüktahtakın İE (2022) Stage-t scenario dominance for risk-averse multi-stage stochastic mixed-integer programs. *Annals of Operations Research* 309:1–35, URL http://dx.doi.org/10.1007/s10479-021-04388-3.

[30] Büyüktahtakın İE, des Bordes E, Kıbış EY (2018) A new epidemics–logistics model: Insights into controlling the ebola virus disease in west africa. *European Journal of Operational Research* 265(3):1046–1063, URL http://dx.doi.org/10.1016/j.ejor.2017.08.037.

[31] Büyüktahtakın İE, Feng Z, Frisvold G, Szidarovszky F, Olsson A (2011) A dynamic model of controlling invasive species. *Computers & Mathematics with Applications* 62(9):3326–3333.

[32] Çetinkaya İO, İ Esra Büyüktahtakın, Shojaee P, Reddy CK (2026) Discovering heuristics with large language models (LLMs) for mixed-integer programs: Single-machine scheduling. *Computers & Operations Research* 186:107325, URL http://dx.doi.org/10.1016/j.cor.2025.107325.

[33] Chen X, Liu J, Yin W (2024) Learning to optimize: A tutorial for continuous and mixed-integer optimization. *Science China Mathematics* 67(6):1191–1262, URL http://dx.doi.org/10.1007/s11425-023-2293-3.

[34] Choi SJ, Cibaku E, Svirsko A, Skipper D, Büyüktahtakın İE (2026) Safety-constrained reinforcement learning for naval warfare searching with an intelligent target. *Refereed Proceedings of the 2026 INFORMS Optimization Society Conference (IOS 2026)* (Atlanta, GA).

[35] Choi SJ, Cooper J, Büyüktahtakın Toy E (2024) A temporal convolutional neural network (TCNN) approach to predicting capacitated lot-sizing solutions. *Proceedings of the 2024 IISE Annual Conference & Expo*, 1–6 (Institute of Industrial and Systems Engineers (IISE)), URL http://dx.doi.org/10.21872/2024IISE_7151.

[36] Choi SJ, Jozani K, Cooper JF, Büyüktahtakın İE (2025) Learning to optimize at scale: A benders decomposition-transfORmers framework for stochastic combinatorial optimization. *NeurIPS 2025 Workshop MLxOR: Mathematical Foundations and Operational Integration of Machine Learning for Uncertainty-Aware Decision-Making*, URL https://openreview.net/forum?id=jVcPvWjrQ5, poster paper, published on OpenReview.

[37] Cibaku E, Büyüktahtakın İE (2026) An adaptive k-means and reinforcement learning (rl) algorithm to effective vaccine distribution. *Computers & Operations Research* 185:107275, URL http://dx.doi.org/10.1016/j.cor.2025.107275.

[38] Cooper JF, Choi SJ, Büyüktahtakın İE (2024) Toward transfORmers: Revolutionizing the solution of mixed integer programs with transformers. *Proceedings of the 2024 Industrial and Systems Engineering Research Conference (ISERC)* (Montreal, Canada), URL http://dx.doi.org/10.48550/arXiv.2402.13380, also available as arXiv:2402.13380.

[39] Coşgun Ö, Büyüktahtakın İE (2018) Stochastic dynamic resource allocation for hiv prevention and treatment: An approximate dynamic programming approach. *Computers & Industrial Engineering* 118:423–439, URL http://dx.doi.org/10.1016/j.cie.2018.01.018.

[40] Dai JG, Gluzman M (2022) Queueing network controls via deep reinforcement learning. *Stochastic Systems* 12(1):30–67, URL http://dx.doi.org/10.1287/stsy.2021.0081.

[41] Delage E, Ye Y (2010) Distributionally robust optimization under moment uncertainty with application to data-driven problems. *Operations Research* 58(3):595–612, URL http://dx.doi.org/10.1287/opre.1090.0741.

[42] Donti PL, Amos B, Kolter JZ (2017) Task-based end-to-end model learning in stochastic optimization. *Advances in Neural Information Processing Systems*, volume 30, 5484–5494, URL http://dx.doi.org/10.48550/arXiv.1710.08005.

[43] Duchi J, Hashimoto T, Namkoong H (2023) Distributionally robust losses for latent covariate mixtures. *Operations Research* 71(2):649–664, URL http://dx.doi.org/10.1287/opre.2022.2363.

[44] Duchi JC, Namkoong H (2021) Learning models with uniform performance via distributionally robust optimization. *The Annals of Statistics* 49(3):1378–1406, URL http://dx.doi.org/10.1214/20-AOS2004.

[45] El Balghiti O, Elmachtoub AN, Grigas P, Tewari A (2023) Generalization bounds in the predict-then-optimize framework. *Mathematics of Operations Research* 48(4):2043–2065, URL http://dx.doi.org/10.1287/moor.2022.1330.

[46] Elmachtoub AN, Grigas P (2022) Smart "predict, then optimize". *Management Science* 68(1):9–26, URL http://dx.doi.org/10.1287/mnsc.2020.3922.

[47] Elman JL (1990) Finding structure in time. *Cognitive Science* 14(2):179–211, URL http://dx.doi.org/10.1016/0364-0213(90)90002-E.

[48] Fan M, Wu Y, Liao T, Cao Z, Guo H, Sartoretti G, Wu G (2023) Deep reinforcement learning for uav routing in the presence of multiple charging stations. *IEEE Transactions on Vehicular Technology* 72(5):5732–5746, URL http://dx.doi.org/10.1109/TVT.2022.3232607.

[49] Fioretto F, Mak TWK, Van Hentenryck P (2020) Predicting ac optimal power flows: Combining deep learning and lagrangian dual methods. *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 630–637, URL http://dx.doi.org/10.1609/aaai.v34i01.5403.

[50] Fischetti M, Jo J (2018) Deep neural networks and mixed integer linear optimization. *Constraints* 23(3):296–309, URL http://dx.doi.org/10.1007/s10601-018-9285-6.

[51] Gal Y, Ghahramani Z (2016) Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, 1050–1059, URL http://dx.doi.org/10.48550/arXiv.1506.02142.

[52] Galande N, Jozani KM, Büyüktahtakın İE (2025) Artificial intelligence in supply chain optimization: A systematic review of machine learning models, methods, and applications. *Optimization Online* 1–66, published online December 8, 2025.

[53] Gasse M, Chételat D, Ferroni N, Charlin L, Lodi A (2019) Exact combinatorial optimization with graph convolutional neural networks. *Advances in Neural Information Processing Systems*, volume 32, 15554–15566, URL http://dx.doi.org/10.48550/arXiv.1906.01629, neurIPS 2019.

[54] Gautron R, Maillard OA, Preux P, Corbeels M, Sabbadin R (2022) Reinforcement learning for crop management support: Review, prospects and challenges. *Computers and Electronics in Agriculture* 200:107182, URL http://dx.doi.org/10.1016/j.compag.2022.107182.

[55] Gers FA, Schmidhuber J, Cummins F (2000) Learning to forget: continual prediction with lstm. *Neural Computation* 12(10):2451–2471, URL http://dx.doi.org/10.1162/089976600300015015.

[56] Gijsbrechts J, Boute RN, Van Mieghem JA, Zhang DJ (2022) 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, URL http://dx.doi.org/10.1287/msom.2021.1064.

[57] Goodfellow I, Bengio Y, Courville A (2016) *Deep Learning* (Cambridge, MA: MIT Press), ISBN 9780262035613, URL https://www.deeplearningbook.org/.

[58] Hamilton WL, Ying R, Leskovec J (2017) Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems* 30, URL http://dx.doi.org/10.48550/arXiv.1706.02216.

[59] Harsha P, Jagmohan A, Kalagnanam J, Quanz B, Singhvi D (2025) Deep policy iteration with integer programming for inventory management. *Manufacturing & Service Operations Management* 27(2):369–388, URL http://dx.doi.org/10.1287/msom.2022.0617.

[60] Hausknecht M, Stone P (2015) Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527* URL http://dx.doi.org/10.48550/arXiv.1507.06527.

[61] Hochreiter S, Schmidhuber J (1997) Long short-term memory. *Neural Computation* 9(8):1735–1780, URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[62] Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. *Neural Networks* 2(5):359–366, URL http://dx.doi.org/10.1016/0893-6080(89)90020-8.

[63] Ivanov D (2023) Intelligent digital twin (idt) for supply chain stress-testing, resilience, and viability. *International Journal of Production Economics* 263:108938, URL http://dx.doi.org/10.1016/j.ijpe.2023.108938.

[64] Ivanov D, Dolgui A (2020) Viability of intertwined supply networks: Extending the supply chain resilience angles towards survivability. a position paper motivated by covid-19 outbreak. *International Journal of Production Research* 58(10):2904–2915, URL http://dx.doi.org/10.1080/00207543.2020.1750727.

[65] Jozani K, Sageer NA, Eldardiry H, Tunc S, Buyuktahtakin Toy E (2025) A multi-echelon demand-driven supply chain model for proactive optimal control of epidemics: Insights from a covid-19 study URL http://dx.doi.org/10.48550/arXiv.2510.16969.

[66] Kaelbling LP, Littman ML, Cassandra AR (1998) Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1–2):99–134, URL http://dx.doi.org/10.1016/S0004-3702(98)00023-X.

[67] Kallus N, Mao X (2023) Stochastic optimization forests. *Management Science* 69(4):1975–1994, URL http://dx.doi.org/10.1287/mnsc.2022.4458.

[68] Kendall A, Gal Y (2017) What uncertainties do we need in bayesian deep learning for computer vision? *Advances in Neural Information Processing Systems*, volume 30, URL http://dx.doi.org/10.48550/arXiv.1703.04977.

[69] Kerr CC, Stuart RM, Mistry D, Abeysuriya RG, Rosenfeld K, Hart GR, Nuñez RC, Cohen JA, Selvaraj P, Hagedorn B, George L, Jastrzebska M, Izzo A, Fowler G, Palmer A, Delport D, Scott N, Kelly S, Bennette CS, Wagner B, Chang ST, Vassall A, Pearson BJ, Winskill PH, Panovska-Griffiths A, Famulare M, Klein DJ (2021) Covasim: An agent-based model of COVID-19 dynamics and interventions. *PLoS Computational Biology* 17(7):e1009149, URL http://dx.doi.org/10.1371/journal.pcbi.1009149.

[70] Khalil EB, Dai H, Zhang Y, Dilkina B, Song L (2017) Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems*, URL http://dx.doi.org/10.48550/arXiv.1704.01665.

[71] Khalil EB, Le Bodic P, Song L, Nemhauser G, Dilkina B (2016) Learning to branch in mixed integer programming. *Proceedings of the AAAI Conference on Artificial Intelligence* 30(1):724–731, URL http://dx.doi.org/10.1609/aaai.v30i1.10080.

[72] Khalil EB, Morris C, Lodi A (2022) Mip-gnn: A data-driven framework for guiding combinatorial solvers. *Proceedings of the AAAI Conference on Artificial Intelligence* 36(9):10219–10227, URL http://dx.doi.org/10.1609/aaai.v36i9.21262.

[73] Kıbış EY, Büyüktahtakın İE (2019) Optimizing multi-modal cancer treatment under 3d spatio-temporal tumor growth. *Mathematical Biosciences* 307:53–69, URL http://dx.doi.org/10.1016/j.mbs.2018.10.004.

[74] Kıbış EY, Büyüktahtakın İE, Haight RG, Akhundov N, Knight K, Flower CE (2021) A multistage stochastic programming approach to the optimal surveillance and control of the emerald ash borer in cities. *INFORMS Journal on Computing* 33(2):808–834, URL http://dx.doi.org/10.1287/ijoc.2020.0963.

[75] Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations*, URL http://dx.doi.org/10.48550/arXiv.1609.02907.

[76] Komorowski M, Celi LA, Badawi O, Gordon AC, Faisal AA (2018) The artificial intelligence clinician learns optimal treatment strategies for sepsis in intensive care. *Nature Medicine* 24:1716–1720, URL http://dx.doi.org/10.1038/s41591-018-0213-5.

[77] Konda VR, Tsitsiklis JN (2000) Actor-critic algorithms. *Advances in Neural Information Processing Systems* 12:1008–1014, URL https://proceedings.neurips.cc/paper/2000/hash/4e6cd95227cb0c280e99a195be5f6615-Abstract.html.

[78] Kool W, van Hoof H, Welling M (2019) Attention, learn to solve routing problems! *International Conference on Learning Representations*, URL http://dx.doi.org/10.48550/arXiv.1803.08475.

[79] Kotary J, Fioretto F, Van Hentenryck P, Wilder B (2021) End-to-end constrained optimization learning: A survey. *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI-21)*, 4475–4482, URL http://dx.doi.org/10.24963/ijcai.2021/610.

[80] Lakshminarayanan B, Pritzel A, Blundell C (2017) Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in Neural Information Processing Systems*, volume 30, URL http://dx.doi.org/10.48550/arXiv.1612.01474.

[81] Lanctot M, Zambaldi V, Gruslys A, Lazaridou A, Tuyls K, Perolat J, Silver D, Graepel T (2017) A unified game-theoretic approach to multiagent reinforcement learning. Guyon I, Luxburg Uv, Bengio S, Wallach H, Fergus R, Vishwanathan SVN, Garnett R, eds., *Advances in Neural Information Processing Systems 30* (Curran Associates, Inc.).

[82] LeCun Y, Bengio Y, Hinton G (2015) Deep learning. *Nature* 521(7553):436–444, URL http://dx.doi.org/10.1038/nature14539.

[83] LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324, URL http://dx.doi.org/10.1109/5.726791.

[84] Lee M, Büyüktahtakın İE (2025) Mathematical formulation of transformer architecture. *Proceedings of the 20th INFORMS Data Mining and Decision Analytics (DMDA) Workshop*, finalist Paper at the Workshop Best Paper Competition.

[85] Libin P, Moonens A, Verstraeten T, Sanjines FRP, Hens N, Lemey P, Nowé A (2021) Deep reinforcement learning for large-scale epidemic control. Dong Y, Ifrim G, Mladenić D, Saunders C, Hoecke SV, eds., *Machine Learning and Knowledge Discovery in Databases. Applied Data Science and Demo Track*, volume 12461 of *Lecture Notes in Computer Science*, 155–170 (Springer), URL http://dx.doi.org/10.1007/978-3-030-67670-4_10.

[86] Liu Z, Khojandi A, Li X, Mohammed A, Davis RL, Kamaleswaran R (2022) A machine learning–enabled partially observable markov decision process framework for early sepsis prediction. *INFORMS Journal on Computing* 34(4):2039–2057, URL http://dx.doi.org/10.1287/ijoc.2022.1176.

[87] Lodi A, Zarpellon G (2017) On learning and branching: A survey. *TOP* 25(2):207–236, URL http://dx.doi.org/10.1007/s11750-017-0451-6.

[88] Lundberg SM, Lee SI (2017) A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, volume 30, 4765–4774, URL http://dx.doi.org/10.48550/arXiv.1705.07874.

[89] Luong MT, Pham H, Manning CD (2015) Effective approaches to attention-based neural machine translation. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1412–1421 (Lisbon, Portugal: Association for Computational Linguistics), URL http://dx.doi.org/10.18653/v1/D15-1166.

[90] Mandi J, Kotary J, Berden S, Mulamba M, Bucarey V, Guns T, Fioretto F (2024) Decision-focused learning: Foundations, state of the art, benchmark and future opportunities. *Journal of Artificial Intelligence Research* 81:1623–1701, URL http://dx.doi.org/10.1613/jair.1.15320.

[91] Mandl C, Minner S (2023) Data-driven optimization for commodity procurement under price uncertainty. *Manufacturing & Service Operations Management* 25(2):371–390, URL http://dx.doi.org/10.1287/msom.2020.0890.

[92] Mazyavkina N, Sviridov S, Ivanov S, Burnaev E (2021) Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research* 134:105400, URL http://dx.doi.org/10.1016/j.cor.2021.105400.

[93] Mišić VV (2020) Optimization of tree ensembles. *Operations Research* 68(5):1605–1624, URL http://dx.doi.org/10.1287/opre.2019.1928.

[94] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D (2015) Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533, URL http://dx.doi.org/10.1038/nature14236.

[95] Mohajerin Esfahani P, Kuhn D (2018) Data-driven distributionally robust optimization using the wasserstein metric: Performance guarantees and tractable reformulations. *Mathematical Programming* 171(1–2):115–166, URL http://dx.doi.org/10.1007/s10107-017-1172-1.

[96] Nakabi TA, Toivanen P (2021) Deep reinforcement learning for energy management in a microgrid with flexible demand. *Sustainable Energy, Grids and Networks* 25:100413, URL http://dx.doi.org/10.1016/j.segan.2020.100413.

[97] Nazari M, Oroojlooy A, Snyder LV, Takáč M (2018) Reinforcement learning for solving the vehicle routing problem. *Advances in Neural Information Processing Systems*, URL http://dx.doi.org/10.48550/arXiv.1802.04240.

[98] Oroojlooyjadid A, Nazari M, Snyder LV, Takáč M (2022) A deep q-network for the beer game: Deep reinforcement learning for inventory optimization. *Manufacturing & Service Operations Management* 24(1):285–304, URL http://dx.doi.org/10.1287/msom.2020.0939.

[99] Pan X, Zhao T, Chen M, Zhang S (2021) Deepopf: A deep neural network approach for security-constrained dc optimal power flow. *IEEE Transactions on Power Systems* 36(3):1725–1735, URL http://dx.doi.org/10.1109/TPWRS.2020.3026379.

[100] Patel RM, Dumouchelle J, Khalil EB, Bodur M (2022) Neur2sp: Neural two-stage stochastic programming. *Advances in Neural Information Processing Systems*, volume 35, 23992–24005, URL http://dx.doi.org/10.48550/arXiv.2205.12006, neurIPS 2022.

[101] Powell WB (2011) *Approximate Dynamic Programming: Solving the Curses of Dimensionality* (Hoboken, NJ: John Wiley & Sons), 2 edition, ISBN 9780470171554, URL http://dx.doi.org/10.1002/9781118029176.

[102] Powell WB (2019) A unified framework for stochastic optimization. *European Journal of Operational Research* 275(3):795–821, URL http://dx.doi.org/10.1016/j.ejor.2018.07.014.

[103] Powell WB (2022) *Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions* (Hoboken, NJ: Wiley), ISBN 9781119815068, URL https://www.wiley.com/en-us/Reinforcement+Learning+and+Stochastic+Optimization%3A+A+Unified+Framework+for+Sequential+Decisions-p-9781119815037.

[104] Puterman ML (1994) *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (New York: John Wiley & Sons), URL http://dx.doi.org/10.1002/9780470316887.

[105] Ribeiro MT, Singh S, Guestrin C (2016) "why should I trust you?": Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1135–1144, URL http://dx.doi.org/10.1145/2939672.2939778.

[106] Rockafellar RT, Wets RJB (1998) *Variational Analysis* (Berlin, Heidelberg: Springer), URL http://dx.doi.org/10.1007/978-3-642-02431-3.

[107] Romera-Paredes B, Barekatain M, Novikov A, Balog M, Kumar MP, Dupont E, Ruiz FJR, Ellenberg JS, Wang P, Fawzi O, Kohli P, Fawzi A (2024) Mathematical discoveries from program search with large language models. *Nature* 625:468–475, URL http://dx.doi.org/10.1038/s41586-023-06924-6.

[108] Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. *Nature* 323(6088):533–536, URL http://dx.doi.org/10.1038/323533a0.

[109] Sadana U, Chenreddy A, Delage E, Forel A, Frejinger E, Vidal T (2025) A survey of contextual optimization methods for decision-making under uncertainty. *European Journal of Operational Research* 320(2):271–289, URL http://dx.doi.org/10.1016/j.ejor.2024.03.020.

[110] Sagawa S, Koh PW, Hashimoto TB, Liang P (2020) Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. *arXiv preprint arXiv:1911.08731* URL http://dx.doi.org/10.48550/arXiv.1911.08731.

[111] Saghafian S (2024) Ambiguous dynamic treatment regimes: A reinforcement learning approach. *Management Science* 70(9):5667–5690, URL http://dx.doi.org/10.1287/mnsc.2022.00883.

[112] Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G (2009) The graph neural network model. *IEEE Transactions on Neural Networks* 20(1):61–80, URL http://dx.doi.org/10.1109/TNN.2008.2005605.

[113] Shapiro A, Dentcheva D, Ruszczyński A (2009) *Lectures on Stochastic Programming: Modeling and Theory* (Philadelphia, PA: Society for Industrial and Applied Mathematics and Mathematical Programming Society), URL http://dx.doi.org/10.1137/1.9780898718751.

[114] Simchi-Levi D, Mellou K, Menache I, Pathuri J (2025) Large language models for supply chain decisions. *SSRN Electronic Journal* URL http://dx.doi.org/10.2139/ssrn.5370043.

[115] Sinha A, Namkoong H, Duchi J (2018) Certifying some distributional robustness with principled adversarial training. *arXiv preprint arXiv:1710.10571* URL http://dx.doi.org/10.48550/arXiv.1710.10571.

[116] Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, volume 27, 3104–3112, URL http://dx.doi.org/10.48550/arXiv.1409.3215.

[117] Sutton RS, Barto AG (2018) *Reinforcement Learning: An Introduction* (Cambridge, MA: MIT Press), 2 edition, ISBN 9780262039246, URL http://incompleteideas.net/book/the-book-2nd.html.

[118] Tong J, Cai J, Serra T (2024) Optimization over trained neural networks: Taking a relaxing walk. *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 221–233, Lecture Notes in Computer Science (Springer), URL http://dx.doi.org/10.1007/978-3-031-60599-4_14.

[119] Vaccaro M, Almaatouq A, Malone T (2024) When combinations of humans and ai are useful: A systematic review and meta-analysis. *Nature Human Behaviour* 8:2293–2303, URL http://dx.doi.org/10.1038/s41562-024-02024-1.

[120] van den Oord A, Dieleman S, Zen H, Simonyan K, Vinyals O, Graves A, Kalchbrenner N, Senior A, Kavukcuoglu K (2016) Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* URL http://dx.doi.org/10.48550/arXiv.1609.03499.

[121] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. *Advances in Neural Information Processing Systems*, volume 30, 5998–6008, URL http://dx.doi.org/10.48550/arXiv.1706.03762.

[122] Venkatachalam S (2025) Integrating large language models with network optimization for interactive and explainable supply chain planning: A real-world case study. *arXiv preprint arXiv:2508.21622* URL http://dx.doi.org/10.48550/arXiv.2508.21622.

[123] Vinyals O, Fortunato M, Jaitly N (2015) Pointer networks. *Advances in Neural Information Processing Systems* 28, URL http://dx.doi.org/10.48550/arXiv.1506.03134.

[124] Watkins CJCH (1989) *Learning from Delayed Rewards*. Ph.D. thesis, University of Cambridge, URL https://www.cs.utexas.edu/~shivaram/readings/b2hd-Watkins1989.html.

[125] Watkins CJCH, Dayan P (1992) Q-learning. *Machine Learning* 8(3–4):279–292, URL http://dx.doi.org/10.1007/BF00992698.

[126] Watson JP, Woodruff DL (2011) Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. *Computational Management Science* 8(4):355–370, URL http://dx.doi.org/10.1007/s10287-010-0125-4.

[127] Werbos PJ (1990) Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE* 78(10):1550–1560, URL http://dx.doi.org/10.1109/5.58337.

[128] Wiesemann W, Kuhn D, Sim M (2014) Distributionally robust convex optimization. *Operations Research* 62(6):1358–1376, URL http://dx.doi.org/10.1287/opre.2014.1314.

[129] Wilder B, Dilkina B, Tambe M (2019) Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 1658–1665, URL http://dx.doi.org/10.1609/aaai.v33i01.33011658.

[130] Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8(3–4):229–256, URL http://dx.doi.org/10.1007/BF00992696.

[131] Wu G, Fan M, Shi J, Feng Y (2023) Reinforcement learning based truck-and-drone coordinated delivery. *IEEE Transactions on Artificial Intelligence* 4(4):754–763, URL http://dx.doi.org/10.1109/TAI.2021.3087666.

[132] Wu Z, Pan S, Chen F, Long G, Zhang C, Yu PS (2021) A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* 32(1):4–24, URL http://dx.doi.org/10.1109/TNNLS.2020.2978386.

[133] Xu H, Mannor S (2012) Distributionally robust markov decision processes. *Mathematics of Operations Research* 37(2):288–300, URL http://dx.doi.org/10.1287/moor.1120.0540.

[134] Yang CL, Bauer K, Li X, Hinz O (2025) My advisor, her ai, and me: Evidence from a field experiment on human–ai collaboration and investment decisions. *Management Science* 72(1):242–264, URL http://dx.doi.org/10.1287/mnsc.2022.03918.

[135] Yilmaz D, Büyüktahtakın İE (2024) A deep reinforcement learning framework for solving two-stage stochastic programs. *Optimization Letters* 18:1993–2020, URL http://dx.doi.org/10.1007/s11590-023-02009-5.

[136] Yilmaz D, Büyüktahtakın İE (2025) A non-anticipative learning-optimization framework for solving multi-stage stochastic programs. *Annals of Operations Research* 355(3):2859–2899, URL http://dx.doi.org/10.1007/s10479-024-06100-7.

[137] Yilmaz D, İ Esra Büyüktahtakın (2023) Learning optimal solutions via an LSTM-optimization framework. *Operations Research Forum* 4(2):48, URL http://dx.doi.org/10.1007/s43069-023-00224-5.

[138] Yilmaz D, İ Esra Büyüktahtakın (2024) An expandable machine learning-optimization framework for sequential decision-making. *European Journal of Operational Research* 314(1):280–296, URL http://dx.doi.org/10.1016/j.ejor.2023.10.045.

[139] Yin X, Büyüktahtakın İE (2021) A multi-stage stochastic programming approach to epidemic resource allocation with equity considerations. *Health Care Management Science* 24(3):597–622, URL http://dx.doi.org/10.1007/s10729-021-09559-z.

[140] Yin X, Büyüktahtakın İE, Patel BP (2023) COVID-19: Data-driven optimal allocation of ventilator supply under uncertainty and risk. *European Journal of Operational Research* 304(1):255–275, URL http://dx.doi.org/10.1016/j.ejor.2021.11.052.

[141] Zou J, Ahmed S, Sun XA (2019) Stochastic dual dynamic integer programming. *Mathematical Programming* 175(1):461–502, URL http://dx.doi.org/10.1007/s10107-018-1249-5.