On the Impact of Deep Learning-based Time-series Forecasts on Multistage Stochastic Programming Policies

Juyoung Wang [a], Mucahit Cevik [b*], and Merve Bodur [a]

[a]Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, ON, Canada; [b]Department of Mechanical and Industrial Engineering, Ryerson University, Toronto, ON, Canada

ABSTRACT

Multistage stochastic programming provides a modeling framework for sequential decision-making problems that involve uncertainty. One typically overlooked aspect of this methodology is how uncertainty is incorporated into modeling. Traditionally, statistical forecasting techniques with simple forms, e.g., (first-order) autoregressive time-series models, are used to extract scenarios to be added to optimization models to represent the uncertain future. However, often times, the performance of these forecasting models are not thoroughly assessed. Motivated by the advances in probabilistic forecasting, we incorporate a deep learning-based time-series forecasting method into multistage stochastic programming framework, and compare it with the cases where a traditional forecasting method is employed to model the uncertainty. We assess the impact of more accurate forecasts on the quality of two commonly used look-ahead policies, a deterministic one and a two-stage one, in a rolling-horizon framework on a practical problem. Our results illustrate that more accurate forecasts contribute substantially to the model performance, and enable obtaining high-quality solutions even from computationally cheap heuristics. They also show that the probabilistic forecasting capabilities of deep learning-based methods can be especially beneficial when used as a (conditional) sampling tool for scenario-based models, and to predict the worst-case scenario for risk-averse models.

KEYWORDS

Multistage stochastic programming; Policy evaluation; Lot-sizing problem; Time-series forecasting; Deep learning; Autoregressive process

1. Introduction

The vast majority of mathematical programming applications assumes deterministic, static data. However, real world problems almost always include some *uncertain parameters*. One such problem is supply chain management where product demand is highly uncertain, while electric power generation problem is another example where energy demand, water/wind inflow are possible uncertain factors at the time of generation. It has been traditionally difficult to predict such uncertainties with high accuracy, however, with the existence of substantial historical data and advances in big data analytics, it is becoming possible to model uncertainty by fitting accurate probability distributions over uncertain parameters. A critical part of being able to exploit the available data is to be able to model optimization problems by taking uncertainty into account, which is the case in *stochastic programming*. Such models provide very

*Corresponding author. Email: mcevik@ryerson.ca

useful information to decision-makers, yielding solutions that are hedged against future uncertainty. Stochastic programming has numerous applications in areas such as scheduling, production planning, supply chain management, energy, finance, manufacturing, healthcare, and natural resources (Wallace and Ziemba 2005).

The most widely applied stochastic programming models involve two main decision stages. In *two-stage stochastic programs*, the first-stage decisions have to be made before observing random outcomes of uncertain parameters, while the second-stage decisions are made after all the uncertainty has been revealed. However, in the majority of practical problems, the planning horizon has more than two decision stages and the uncertainty is revealed gradually over time.

Multistage stochastic programming provides a modeling framework for sequential decision-making problems under uncertainty. In the classical setting, the uncertainty information is represented by a stochastic process with a given probability distribution and support, and the goal is to find a *policy*, that defines decisions to be made at each decision stage, while optimizing an objective function of an expected value form. More specifically, the decisions at each stage are functions of the observed outcomes up to that stage, which should satisfy a given set of constraints almost surely with respect to the distribution of the stochastic process; and the objective is to minimize/maximize the total expected cost/profit of the feasible policies over all stages. Some risk measures other than the expected value, such as variance and conditional value at risk, can be incorporated into stochastic programs to obtain risk-averse variants, one of the most conservative one being robust optimization models that optimize the worst-case cost/profit. In this paper, we focus on risk-neutral multistage stochastic programs (MSPs), while providing one set of experiments using the robust optimization variant.

The fundamental assumption of stochastic programming is that the *probability distribution* of the underlying stochastic process is given. Thus, the process of fitting an accurate (joint) probability distribution over uncertain parameters of a given decision-making problem is of utmost importance to obtain high-quality policies from a stochastic program. However, this prior process is usually glossed over in stochastic programming applications. Also considering the computational difficulties associated with stochastic programs, in particular due to the famous curse of dimensionality which states the exponential growth in the problem size in terms of the number of decision stages, many crude simplifying assumptions are made for MSPs. In the literature, random variables associated with different stages are commonly assumed to be independent (referred to as stage-wise independence), and standard distribution of simple form (e.g., uniform, Gaussian, and lognormal) are fitted for random variables. In order to preserve some dependence between stages, simple (e.g., first-order) autoregressive time-series models are mostly adopted.

Although these assumptions provide a great deal of help to relieve the computational burden for solution methods, especially for the ones that require the construction of scenario trees such as stochastic dual dynamic programming (Pereira and Pinto 1991), they can be deemed inadequate to incorporate stochastic processes with high order of dependency due to the explosive state-space expansion, and in such a case, they can be even computationally intractable. In fact it is mostly the case that the stage-wise independence assumption is violated in practice, and the underlying probability distribution cannot be represented well with a simple form, notably as a reasonable sized scenario tree. On the other hand, *deep learning-based time-series forecasting* methods can provide highly accurate models for uncertainty. Moreover, when neural network-based methods (e.g., recurrent neural networks and convolutional neural networks) are used, stage-wise dependencies can be modeled inherently. Those promising models of

uncertainty can then be combined with some scenario tree-free solution methods, such as deterministic or two-stage stochastic programming approximations, and decision rule-based approaches, which allow users to solve the optimization problem without suffering from the curse of dimensionality caused by the exponential growth of the scenario tree. As one can expect, depending on the quality of forecasts, simpler methods as in the former class can yield high-quality solutions for MSPs, which is the focus of our computational study.

In this paper, we conduct an empirical study on the importance of the assumed probability distribution for MSPs by analyzing the impact of better forecasts. More specifically, we compare the quality of certain look-ahead policies under the assumption of different uncertainty models. Look-ahead policies are commonly used in the context of sequential optimization problems. The term "look-ahead" indicates that the impact of current stage decisions on the future ones are taken into account while making decisions. In the realm of sequential decision making under uncertainty, future uncertainty is also considered. There are several classes of look-ahead policies, which mainly differ in the complexity of the optimization problems and the type of forecasts for uncertain parameters. In our work, we consider two most commonly used look-ahead policies: (i) a deterministic policy that relies on the conditional expected scenario for the future, and (ii) a two-stage policy that approximates the future uncertain stages by means of a finite set of scenarios.

We consider two commonly used primal look-ahead policies for MSPs in a rolling-horizon framework: (i) a deterministic policy that relies on the conditional expected scenario for the future, and (ii) a two-stage policy that approximates the future uncertain stages by means of a finite set of scenarios. For the scenario generation phase of these rolling-horizon schemes, we adopt a modern deep learning-based time-series forecasting method, namely DeepAR (Salinas et al. 2019), and compare it with the case where a traditional time-series model, namely first-order autoregressive (AR) model or moving average type model, is used. In our numerical experiments on a multi-item stochastic lot-sizing problem, using two different well-known demand datasets, we find that the use of improved forecasting methods lead to significantly better policies. In particular, compared to AR(1), DeepAR-based solutions yield 16% improvement on average under the risk-neutral setting based on our performance measure — the percentage gap of the obtained objective value to the perfect information bound, where the perfect information bound is defined as the expected value of the optimal values of deterministic optimization problems obtained by individual scenarios (i.e., particular realizations of the random variables in the model). Moreover, in a risk-averse setting, which rely on using the prediction of the worst-case realization, DeepAR-based solutions lead to 56% improvement on average over AR(1)-based solutions, emphasizing the enhanced probabilistic forecasting capabilities of DeepAR. Lastly, in the risk-neutral setting, overall, we find that two-stage policies outperform deterministic ones, and most notably DeepAR-based two-stage policies perform the best, for instance yielding 20% improvement on the average over the AR(1)-based two-stage policies, which highlights the sampling power of DeepAR over traditional methods. All these findings reveal how helpful the use of deep learning-based forecasts can be in the realm of stochastic programming and robust optimization.

To the best of our knowledge, this is the first study incorporating modern deep learning-based time-series forecasting methods into multistage stochastic programming framework. Specifically, we make use of a recent deep-learning based method that is specifically designed to provide high-accuracy probabilistic forecasts, in arguably the most relevant context, as a sampling tool in the (multistage) stochastic

optimization. Also, we illustrate its benefits in a practical setting, considering a very common application of multistage stochastic programming in supply chain management, and using real demand datasets. As such, our study contributes to bridging the gap between machine learning and operations research.

The remainder of this paper is organized as follows. In Section 2, we review the relevant literature about MSP solution methods and time-series forecasting in the realm of stochastic programming. In Section 3, we provide an overview of MSPs, where the necessary notation for the rest of the paper is also introduced. We present the considered policies for MSPs and the rolling-horizon framework in Section 4, and the forecasting processes for scenario generation in Section 5. We describe our computational study setup in Section 6, and report our numerical results in Section 7. Finally, we summarize our findings and conclude the paper in Section 8.

2. Literature review

Despite their expressive ability in modeling various real-life problems, MSPs have not been widely used in practice as they are notoriously difficult to solve. The major algorithmic challenges stem from the lack of nice properties such as convexity and continuity of the so-called value function in the existence of decisions taking only integer values, and the difficulty in computing the expectation in the objective function, especially when uncertain parameters have continuous distributions. As such, the majority of the literature considers the case where all decision variables are continuous, and proposes approximate solution methods, following one of the two approaches: Simplifying the underlying stochastic process to a scenario tree, and restricting the functional form of the policies. For the former case, there are a variety of available methods such as nested Benders decomposition (Birge 1985b; Pereira and Pinto 1991), aggregation and partitioning (Bakir et al. 2020; Birge 1985a), and progressive hedging (Watson and Woodruff 2011). In the latter approach, the complexity of the problem is reduced by enforcing decisions made at each stage to be a specific function of the observed outcomes up to that stage. Different functional forms, also known as decision rules, are proposed in the literature such as linear (Shapiro and Nemirovski 2005), piecewise linear (Chen et al. 2008), polynomial (Bampou and Kuhn 2011) and two-stage (Bodur and Luedtke 2018). On the other hand, the literature is rather limited for MSPs involving integer variables. Regarding the scenario tree-based approaches, there exist some generic decomposition methods (CarøE and Schultz 1999; Lulli and Sen 2004), as well as some algorithms designed for MSPs with binary and continuous variables only (Alonso-Ayuso et al. 2003; Zou et al. 2019). For decision rules, the standard approach for the pure continuous case is extended to the binary case via piecewise linear binary functions as the policy form (Bertsimas and Georghiou 2015), and more recently Lagrangian dual decision rules are proposed for multistage stochastic mixed integer programs (Daryalal et al. 2020).

Time-series forecasting methods are designed to take spatio-temporal relations into account. Earlier studies on time-series forecasting focus on linear prediction models such as autoregressive (AR), moving average (MA) and auto-regressive integrated moving average (ARIMA) models where a linear function of past observations is used to predict the future values (Box et al. 2015). Recent advances in artificial neural networks and deep learning allow practitioners to use deep learning in time-series forecasting, which provides the ability to process a large amount of data (e.g., a larger portion of the temporal data). Long short-term memory neural networks and gated

recurrent units are among the most commonly used deep learning approaches for time-series forecasting with immediate applications in various fields such as power systems and marketing (Agrawal et al. 2018; Cheng et al. 2017; Kuan et al. 2017). Recent deep learning-based time-series prediction studies tend to involve more advanced neural network architectures such as DeepAR (Salinas et al. 2019) and Deep State Space (Rangapuram et al. 2018) models. Unlike traditional time series forecasting models, such deep learning-based models leverage information from multiple time series to yield improved forecasts for any target time series. We refer readers to recent review papers (Fawaz et al. 2019; Gamboa 2017) on deep learning methods in time-series prediction for a more detailed list of relevant studies.

In order to overcome some computational difficulties associated with MSPs, that are exacerbated when there are many uncertain parameters in the problem, uncertainty models of a simple form have been mostly used in the literature. Under the common independence assumption, a model is built for each random variable, mostly in the form of a uniform, normal or lognormal distribution. In a more complex setting where the stage-wise independence assumption has been relaxed, previous studies mostly rely on low-order autoregressive models, namely AR(1) and AR(2), to capture some dependencies in uncertain parameters, as in (Daryalal et al. 2020; Shapiro et al. 2014).

However, the accuracy of the forecasts might have a significant impact on the MSP model results, for both the actual quality of obtained solutions and accuracy of used performance measures, as well as on the number of scenarios required to obtain stable objective function values, thus on the complexity of the solution approach.

As mentioned above, accurate inclusion of uncertainty to optimization models is a task of great importance. Some recent works address this issue and discuss the value of utilizing a precise representation of uncertainty by incorporating time-series modeling techniques relatively complex compared to the traditionally used AR(1) model. For instance, a robust optimization model for hydro-electric reservoir management problems is proposed in (Gauvin et al. 2018a), where the uncertainty set is built with the help of ARMA and GARCH models. It is also proven that ARMA model can be used to build an ellipsoidal type of uncertainty set to model the uncertainty of water inflow to the hydro-electric reservoir. Due to the stage-wise dependence of used time-series models, stochastic dual dynamic programming could not be used, instead affine decision rules are used to derive approximate solutions. The numerical studies in (Gauvin et al. 2018a) indeed provide evidence of acquirable profit in employing sophisticated uncertainty modeling approaches, instead of simpler ones. In (Gauvin et al. 2018b), a nonlinear time-series uncertainty set modeling approach is proposed for the same problem class of (Gauvin et al. 2018a). Unlike the model proposed in (Gauvin et al. 2018a), the introduced nonlinearity of time-series makes the model non-convex. To cope with such an issue, a specialized successive linear programming algorithm is proposed. For a comprehensive review of how uncertainties are modeled in the context of energy generation planning, we refer readers to (Lorca et al. 2020). To the best of our knowledge, modern deep learning-based time-series forecasting techniques have not been incorporated into stochastic programming.

Quantitative studies on the benefit of precisely modeling uncertainty are performed also in contexts different than energy generation planning, such as call center demand prediction. In (Ye et al. 2019), a time-series model is developed, and it is shown that under some dependence condition of call center demand streams, performing a joint forecast is more beneficial than building a single prediction model for each stream separately. More specifically, three types of inter-stream dependence are considered and modeled simultaneously, as well as certain data conditions where the multi-stream

approach performs better than the single stream model are provided. This finding also highlights the importance of taking dependencies into account while modeling uncertainty. In our approach, we use DeepAR which considers such dependencies by design while building a model for uncertainty.

3. Multistage stochastic programs

In this section, we introduce a generic multistage stochastic programming formulation, mostly following the presentation of (Shapiro et al. 2014, Chapter 3). A generic T-stage stochastic programming problem, driven by an exogenous random data (i.e., stochastic) process, can be modeled as follows:

$$\min \quad \mathbb{E}_{\xi_{[T]}} \left[\sum_{t \in \mathcal{T}} f_t \left(x_t(\xi_{[t]}), \xi_t \right) \right] \tag{1a}$$

$$\text{s.t.} \quad x_t(\xi_{[t]}) \in \mathcal{X}_t \left(x_{t-1}(\xi_{[t-1]}), \xi_t \right), t \in \mathcal{T}, \text{w.p.1} \tag{1b}$$

where

- $\mathcal{T} = \{1, 2, \ldots, T\}$ is the set of decision *stages*.
- ξ_t is the vector of random variables at stage $t \in \mathcal{T}$.
- $\{\xi_t\}_{t \in \mathcal{T}}$ represents the underlying stochastic process.
- $\xi_{[t]} = (\xi_1, \xi_2, \ldots, \xi_t)$ is the history of the stochastic process up to stage $t \in \mathcal{T}$.
- $\{x_t(\xi_{[t]})\}_{t \in \mathcal{T}}$ describe a *policy*, a solution to the MSP. Due to the underlying randomness, they are functions of the stochastic process. However, they are *nonanticipative*, i.e., only depend on the history of the process up to the stage they are to be made.
- It is assumed that $\xi_1 = 1$, i.e., the first stage is *deterministic*.
- $x_0(\xi_0)$ represents the initial state of the system.
- $f_t(\cdot, \cdot)$ is the cost function associated with stage $t \in \mathcal{T}$; it maps all the decisions made up to stage t, most notably the stage-t decisions, and the lastly observed random outcomes (i.e., the realization of ξ_t) into a real value.
- $\mathcal{X}_t(\cdot, \cdot)$ describes the feasible set for decisions to be made at stage $t \in \mathcal{T}$; it maps all the decisions made up to the previous stage, $t-1$, and the lastly observed outcomes, of the random vector ξ_t, into a set of real-valued vectors.

This modeling framework is illustrated in Figure 1.

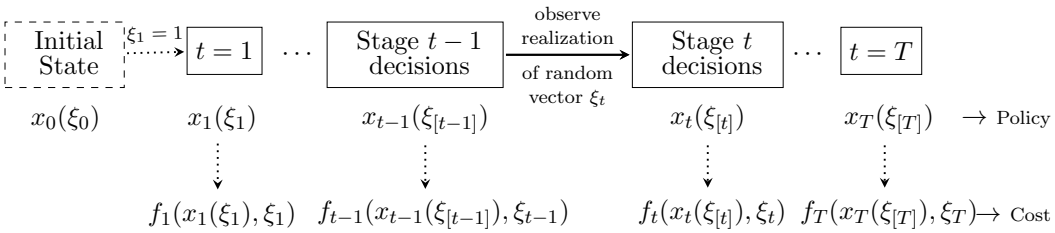


Figure 1. Multistage stochastic programming framework.

The objective function (1a) minimizes the expected total cost of the selected policy over all decision stages. Note that the expectation is taken over the (joint) distribution

of the full stochastic process. Constraints (1b) ensure that a feasible policy is selected, i.e., the functional decision variables at every stage satisfy the associated constraints with probability one (denoted by w.p.1). We note that some decision variables might be integer, which is also enforced by (1b).

As in general, we assume that the model (1)

- is feasible and has an optimal solution (policy), and
- has relatively complete recourse, i.e., for any stage $t \in \mathcal{T}$, given any observation of the history up to the current stage, $\xi_{[t]}$, and any feasible set of decisions up to the previous stage, $\{x_{t'-1}(\xi_{[t'-1]})\}_{t' \in \{1,\dots,t-1\}}$, there exists a feasible set of decisions for the current stage, $x_t(\xi_{[t]})$.

Lastly, we note that although the solution to an MSP is a policy (which describes the decisions to be made at each stage as a function of the observed history up to that stage), the only *implementable decisions* are the first-stage ones, i.e., $x_1(\xi_1)$, as only the first stage is deterministic (since $\xi_1 = 1$ is a constant, not a random variable); all the other decision variables are random variables themselves.

4. Policies and rolling-horizon framework

The model (1) constitutes an infinite-dimensional optimization problem as both the decision variables and constraints are of infinite size, as both of them are function of random variables with possibly infinite support. Therefore, it is both theoretically and computationally very challenging to derive solutions for it, even in the case where all the objective and constraint functions are linear. The existence of integer variables adds another level of difficulty. Therefore, some heuristic procedures are preferred to construct feasible policies, or rather implementable decisions (i.e., the first-stage solution). More specifically, candidate first-stage solutions are usually obtained by solving a restriction or a relaxation of the original multistage model. For instance,

- a one-stage relaxation can be obtained by ignoring all the future stages, i.e., by removing all the variables $\{x_t(\xi_{[t]})\}_{t \in \mathcal{T} \setminus \{1\}}$ and the constraints associated with them from the model, and solving the remaining single-stage deterministic model;
- a one-stage restriction of the multistage model can be obtained by making all the decision variables deterministic, i.e., for all $t \in \mathcal{T} \setminus \{1\}$, by replacing $x_t(\xi_{[t]})$ with $x_t(\xi_1)$, taking away their flexibility to depend on the observed history; or by applying static linear decision rules (Shapiro and Nemirovski 2005);
- a two-stage relaxation can be obtained by relaxing the nonanticipativity constraints on all the decision variables except the first-stage ones, i.e., for all $t \in \mathcal{T} \setminus \{1\}$, by replacing $x_t(\xi_{[t]})$ with $x_t(\xi_{[T]})$ to make them functions of the full history (up to T) despite the fact that it will not be available at stage t; and
- a two-stage restriction can be obtained by applying two-stage linear decision rules (Bodur and Luedtke 2018).

Then, in practice, a full implementable solution for a T-stage problem can be obtained by iteratively applying the selected restriction or relaxation technique at every stage, i.e., by applying the chosen first-stage solution generation method in a *rolling-horizon framework*, which is illustrated in Algorithm 1. Rolling-horizon evaluation framework is commonly used for discrete-time sequential decision-making problems under uncertainty (Silvente et al. 2018; Bischi et al. 2019; Daryalal et al. 2020).

Algorithm 1: Rolling-horizon framework

Input: N_S, the number of stages, and N_P, the number of periods ($N_P \geq N_S$), $\hat{x}_0$, vector representing the initial state of the system

Output: An implemented policy $\{\hat{x}_t\}_{t\in\mathcal{T}}$

for $t = 1, 2, \ldots, T$ **do**
- $\hat{\xi}_t \leftarrow$ observed ξ_t realization
- $\mathcal{P} = \{t, t+1, \ldots, \min\{T, t+N_P-1\}\}$
- **if** $t > T - N_S + 1$ **then**
  - $N_S = T - t + 1$
- $\{\mathcal{P}_n\}_{n=1,2,\ldots,N_S} \leftarrow$ **PartitionPeriods**$(\mathcal{P}, N_S)$
- $\hat{x}_t \leftarrow$ **Optimize**$(\hat{x}_{t-1}, \hat{\xi}_{[t]}, \{\mathcal{P}_n\}_{n=1,2,\ldots,N_S})$

In Algorithm 1, a policy can be constructed gradually by solving a sequence of optimization problems considering N_P consecutive stages ahead, each being referred to as a *period*, but grouping them into N_S decision *stages* (which is usually small such as one or two). In the algorithm,

- PartitionPeriods$(\cdot)$ function takes the list of considered periods and partitions them into N_S sets such that (i) each partition set consists of consecutive periods, and (ii) the current stage t belongs to the first partition set $\mathcal{P}_1$; while
- Optimize$(\cdot)$ function builds and solves an N_S-stage stochastic program where stage n consists of periods $\mathcal{P}_n$, where the initial state is fixed to $\hat{x}_{t-1}$, and returns the optimal first-stage solution.

For instance, Figure 2 illustrates the $N_S = 3$-stage stochastic model to be solved at $t = 3$ considering the $N_P = 5$ periods ahead, $\mathcal{P} = \{3, 4, 5, 6, 7\}$, partitioned as $\mathcal{P}_1 = \{3\}, \mathcal{P}_2 = \{4, 5\}, \mathcal{P}_3 = \{6, 7\}$ for a $T = 8$-stage problem in Algorithm 1.

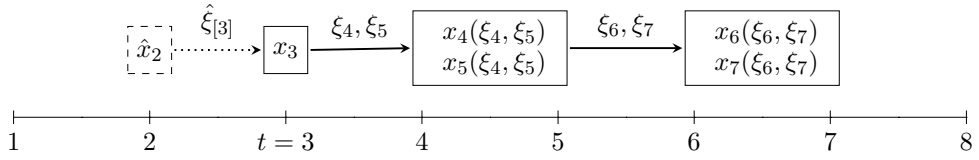


Figure 2. Rolling horizon framework.

In our experiments, we consider all the future stages ($N_P = T$) at any stage, and use a deterministic policy ($N_S = 1$, $\mathcal{P}_1 = \mathcal{P}$) as well as a two-stage stochastic policy ($N_S = 2, \mathcal{P}_1 = \{1\}, \mathcal{P}_2 = \mathcal{P} \setminus \{1\}$).

Next, we provide the details of the Optimize function for the considered policies, namely the associated mathematical models and how to solve them in conjunction with forecasting/sampling methods. We let $\mathbb{P}$ denote the joint probability distribution of all random variables involved in the stochastic process $\{\xi_t\}_{t\in\mathcal{T}}$.

4.1. Deterministic policy

For deterministic policies, at stage $t \in \mathcal{T}$ of the rolling-horizon framework, given the previous stage optimal first-stage solution $\hat{x}_{t-1}$ as the initial state, and the history of

observations $\hat{\xi}_{[t]}$, we solve the following $(T-t+1)$-period deterministic problem:

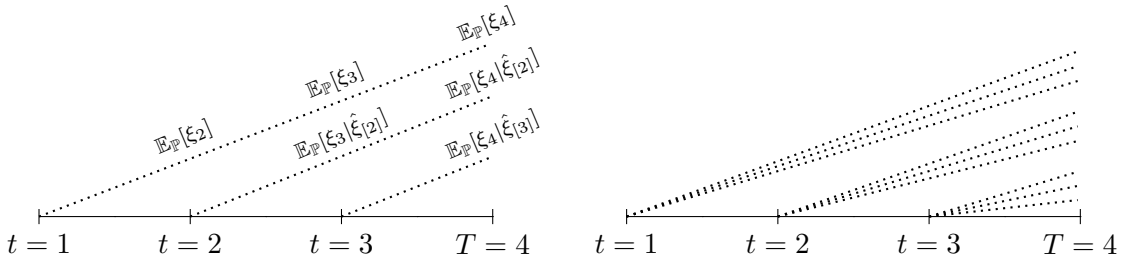
$$\begin{aligned} \min \quad & f_t\left(x_t, \hat{\xi}_t\right) + \sum_{t'=t+1}^{T} f_{t'}\left(x_{t'}, \xi_{t'}^{\text{expected}}\right) \\ \text{s.t.} \quad & x_t \in \mathcal{X}_t\left(\hat{x}_{t-1}, \hat{\xi}_t\right) \\ & x_{t'} \in \mathcal{X}_{t'}\left(x_{t'-1}, \xi_{t'}^{\text{expected}}\right), \; t' = t+1, \ldots, T \end{aligned}$$

where for each period $t' = t+1, \ldots, T$, the expected realization of the associated random vector is obtained by computing its expectation conditioned on the observed history:

$$\xi_{t'}^{\text{expected}} := \mathbb{E}_{\mathbb{P}}[\xi_{t'} | \hat{\xi}_{[t]}] \tag{2}$$

In other words, in this policy, we replace the future by the single conditional expected scenario.

An illustration for this policy is provided in Figure 3a. As shown in the figure, the deterministic policy does not rely on multiple sample paths to describe the uncertain nature of the problem. Instead, at each decision-making stage, the uncertain future is represented by a single sample path obtained by using the conditional mean values of the random variables. As a result, such a policy is computationally cheap. However, as the uncertain factors are replaced by a single future scenario, the variance of the stochastic process is not taken into account. To address such an issue, a two-stage policy that approximates future uncertainty by a set of scenarios, as illustrated in Figure 3b, can be considered as an alternative, which is explained in more detail next.



(a) Deterministic policy.

(b) Two-stage policy.

Figure 3. Look-ahead policies.

4.2. Two-stage policy

In the case of two-stage policies, the future is represented by a finite set of scenarios rather than a single scenario. Fix a stage $t \in \mathcal{T}$ of the rolling-horizon framework.

$$\{(\xi_{t+1}^s, \ldots, \xi_T^s)\}_{s \in \mathcal{S}} \tag{3}$$

be an independent and identically distributed sample of future scenarios generated using the distribution $\mathbb{P}$ conditioned on the observed history $\hat{\xi}_{[t]}$, where $\mathcal{S}$ is the set of

scenarios and s is the scenario index. Then, we solve the following two-stage stochastic program:

$$\begin{aligned} \min \quad & f_t\left(x_t, \hat{\xi}_t\right) + \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \sum_{t'=t+1}^{T} f_{t'}\left(x_{t'}^s, \xi_{t'}^s\right) \\ \text{s.t.} \quad & x_t \in \mathcal{X}_t\left(\hat{x}_{t-1}, \hat{\xi}_t\right) \\ & x_{t'}^s \in \mathcal{X}_{t'}\left(x_{t'-1}^s, \xi_{t'}^s\right), \; t' = t+1, \ldots, T, s \in \mathcal{S} \end{aligned}$$

where the objective function minimizes the average total cost over the future scenarios. Unlike the only implementable non-anticipative decisions x_t, we generate the scenario copies of future decisions, $x_{t'}^s$. As this model does not scale well computationally with the sample size, we usually represent the future using a small set of scenarios.

5. Forecasting for scenario generation

In this section, we describe our proposed approach, followed by the review of forecasting methods employed in our analysis.

5.1. Proposed approach

As the future is represented by only a single scenario, the deterministic approach usually provides a very crude approximation of the underlying MSP, thus yields poor-quality policies. Although the two-stage policy overcomes this issue in theory, it may not be a viable approach in practice either, as it usually requires a large number of scenarios to get good-quality solutions, thus possibly a prohibitive computational effort. Furthermore, if the assumed probability distribution $\mathbb{P}$ is far from the true distribution, one can not expect the two-stage policy perform well even if a large number of scenarios have been considered. Therefore, predicting $\mathbb{P}$ accurately is of paramount importance to obtain high-quality policies. Inspired by the success of recent deep learning-based time-series forecasting methods (Gamboa 2017; Rangapuram et al. 2018; Salinas et al. 2019), we propose to estimate $\mathbb{P}$ by such a modern probabilistic forecasting method, to be given as an input to an MSP. In Section 7, we showcase the impact of improved forecasting methods on MSP policy generation via our empirical study.

5.2. Forecasting methodologies

We compare the quality of the deterministic and two-stage policies using three forecasting methods: AR(1), (log) moving average and DeepAR. We use (log) moving average as a middle-level method in the sense that it is sophisticated enough compared to AR(1), while it is still primitive compared to state-of-the-art deep learning-based time-series forecasting methodologies, such as DeepAR (Salinas et al. 2019) and Deep State Space (Rangapuram et al. 2018) models, among others. Also, MA-based model has been shown to perform well for one of the datasets that we use in our experiments.

5.2.1. AR(1)

AR(1) is one of the most traditionally used forecasting methods in the realm of stochastic programming, due to its theoretical suitability to be applied in many existing MSP solution methodologies. In this modeling scheme, stochastic process $\{\xi_t\}_{t\in\mathcal{T}}$ is assumed to have the form

$$\xi_t = \phi\xi_{t-1} + (1-\phi)\gamma + \varepsilon_t, \text{ for all } t = 2,\dots,T$$

where ϕ is the AR(1) coefficient, γ is the trend constant vector, and ε_t is assumed to be a white noise vector. This model is widely used in the stochastic dual dynamic programming (Pereira and Pinto 1991) literature, due to its suitability for state-space expansion. For further details regarding the use of AR(1) model in the MSP context, we refer readers to (Dowson and Kapelevich 2020; Shapiro 2011).

5.2.2. Autoregressive moving average (ARMA)

ARMA, parametrized by two natural numbers p and q, is commonly used (Gauvin et al. 2018a; Shapiro et al. 2013) forecasting methods that linearly models serial dependency of discrete time dependent sequence of random variables (Box et al. 2015), where the randomness is represented by a white noise term. A stochastic process $\{\xi_t\}_{t\in\mathcal{T}}$ following ARMA(p,q) can be written in the following manner:

$$\xi_t = c + \varepsilon_t + \sum_{i=1}^{p}\varphi_i\xi_{t-i} + \sum_{i=1}^{q}\psi_i\varepsilon_{t-i}, \text{ for all } t = 2,\dots,T$$

where $\{\varepsilon_t\}_t$ is a sequence white noise, while $\{\varphi_i\}_i$ and $\{\psi_i\}_i$ are coefficients used to fit the model. The $\xi_{t'}$ and $\varepsilon_{t'}$ values for $t' \leq 0$ are either known non-zero values or assumed to be zero. Note that by restricting $(p,q) = (1,0)$, we recover the AR(1) model.

In our computational study, for one of the datasets, we use a logarithmic moving average model, which is an ARMA$(0,q)$ model fitted over the time-series obtained by taking the logarithmic transformations of the original time-series observations. We refer readers to (Box et al. 2015) for further details of ARMA.

5.2.3. DeepAR

Before discussing the details of how DeepAR can be used for sampling purposes, we briefly describe the building blocks of DeepAR, assuming familiarity with the basic concepts of neural networks, which can be found for instance in (Goodfellow et al. 2016).

5.2.3.1. Recurrent neural network. Due to the underlying limitation of multilayer neural networks on processing sequential data, a specialized family of neural networks, namely recurrent neural networks (RNNs), has been developed. Let $\{d^t\}_{t\in\mathcal{T}}$ be a network input sequence and assume the data is fed in an ordinal manner. Given a vector Θ of parameters associated with the network and a state mapping function $g(\cdot)$, the state of the hidden units of the network at stage $t \in \mathcal{T}$, defined by $h^t = g(h^{t-1}, d^t; \Theta)$, carries the encoded information of the previous steps by receiving h^{t-1}

as its input. The calculated states are then used for the output generation.

5.2.3.2. DeepAR: probabilistic forecasting with autoregressive recurrent networks. DeepAR is a neural network-based time-series forecasting method developed by Salinas et al. (2019). It uses a specialized RNN architecture, called the long short-term memory, that is capable of carrying and forgetting information. It is a *probabilistic* time-series forecasting method, i.e., its output defines the parameters of a probability distribution. Given history $\hat{\xi}_{[t^*]}$, we assume $i \in \{1, \ldots, N\}$ is used to denote the sample index. In DeepAR, the neural network is trained to maximize the following objective function:

$$\sum_{i=1}^{N} \sum_{t=0}^{t^*} \log(\ell(\hat{\xi}_{i,t} | \theta(\mathbf{h}_{i,t}))$$

where ℓ is the likelihood function, $\mathbf{h}_{i,t}$ represents network hidden layer parameters, and θ is a function that receives network hidden layer parameters and returns parameters for the likelihood function. Due to its construction, it can learn global patterns from many different time-series and covariates, where learned patterns are used to improve the forecasting performance for other time-series. For example, it can be used to predict weekly sales by inferring patterns from covariates such as weekly temperature records, boolean holiday indicators, among others, to improve the overall forecasting performance. Besides DeepAR, there exists other recent deep learning-based probabilistic time series forecasting models (e.g., see Deep-state-space (Rangapuram et al. 2018)). However, we only consider the combination of DeepAR and MSP policies, as DeepAR has been commonly used as a strong benchmark in many studies.

5.2.3.3. Conditional sample acquisition via ancestral sampling with DeepAR. In a rolling-horizon framework, let t^* denote the current stage and $\hat{\xi}_{[t^*]}$ be the observed history. While making the decisions associated with stage t^*, a natural approach is to provide the optimization model with the updated forecasts for the future stages, i.e., the data obtained by retraining the neural network including the newly obtained observations, $\hat{\xi}_{t^*}$. However, such an approach is not practical when training the network requires a significant computational effort, or the allowed time to get updated forecasts between decision-making stages is small. Although there exist many approaches to reduce training times, e.g., transfer learning, retraining might not always be a viable approach for some real-world applications.

To cope with this problem, a heuristic sampling approach can be considered, by exploiting the ancestral sampling capacity of the encoder-decoder architecture of RNN. Ancestral sampling is a commonly used technique in the field of probabilistic graphical models (Goodfellow et al. 2016). It sequentially generates samples from a complex joint distribution in such a way that a single sample at each step is obtained from a relatively easy-to-sample distribution compared to that of the joint density. Such a forecasting process of DeepAR is provided in Algorithm 2, which returns $|\mathcal{S}|$ sample paths for the future, denoted by $\tilde{\xi}^s, s \in \{1, \ldots, |\mathcal{S}|\}$. Note that the algorithm takes as input the current decision-making stage, t^*, and the observed history of the uncertainty, $\hat{\xi}_{[t^*]}$, and generates sample paths without retraining the forecasting model. In other words, while using MSP combined with DeepAR in real-time decision making, future-stage predictions are automatically updated after each observation (i.e., after some uncertainty is revealed to the decision-maker) thanks to the time-step-dependent

forecasting ability of the RNN-based DeepAR architecture.

Algorithm 2: Conditional sample path generation with an RNN-based time-series forecasting method

Input: $h^{t^*-1}, \Theta, \hat{\xi}_{[t^*]}, \mathsf{D}(\cdot)$

Output: $\{\tilde{\xi}_t^s\}_{t\in\{t^*+1,\ldots,T\},s\in\{1,\ldots,|\mathcal{S}|\}}$

for $s \in \{1,\ldots,|\mathcal{S}|\}$ **do**
  Let $\tilde{\xi}_{t^*}^s = \hat{\xi}_{t^*}$
  for $t \in \{t^*,\ldots,T-1\}$ **do**
    $h^t = g(h^{t-1}, \tilde{\xi}_t^s; \Theta)$
    $\tilde{\xi}_{t+1}^s =$ **Sample-from**$(\mathsf{D}(h^t))$

In the algorithm, $\mathsf{D}(h)$ denotes any predefined probability distribution parametrized by h, which in our case is used to refer RNN output parameters. We use **Sample-from**$(\cdot)$ to represent the function that returns a single draw from a given probability distribution. Although this function can be used to incorporate diverse sampling techniques, in this paper, we assume that it uses the crude Monte Carlo sampling.

6. Experimental setup

As previously discussed, there exist many applications that require close coordination of time-series forecasting techniques and MSP. In this work, since our objective is to quantify the performance gap between the combination of general MSP with traditional and modern time-series forecasting techniques, we illustrate the results obtained on a particular MSP application, namely a multi-item lot-sizing problem with backlogging and production lag (Daryalal et al. 2020), which contains both discrete and continuous variables. We conduct experiments using two well-known publicly available sales (demand) datasets.

6.1. Multi-item stochastic lot-sizing problem

The multi-item stochastic lot-sizing problem with backlogging and production lag (MSlag) aims to determine the optimal number of items to be produced at each time period (decision stage) while meeting customer demand for each item at each period, which is described as a stochastic process. Since the production cost at each period might differ and/or future demand may not be satisfied with the existing production capacity due to uncertainty, throughout the planning horizon, the decision maker is allowed to retain items in the inventory, as long as the inventory capacity restriction is satisfied, while spending holding costs for storing products in the inventory. Furthermore, in this problem setup, there exists a non-negative sequence describing fixed setup cost for deciding to produce any product at each period, regardless of the amount of the corresponding items to be produced (Chen et al. 2010). In terms of time taken at production, there is a per period capacity, however overtime is allowed at the expense of incurring a higher production cost. Unmet demand at a period can be backlogged and satisfied at a future period.

The MSP model for MSlag is provided below, and the detailed description for the parameters and decision variables are provided in Table 1.

- **Objective function. Minimize inventory, backlog, production setup, overtime costs**:

$$\min \mathbb{E}\left[\sum_{t\in\mathcal{T}}\left(\sum_{j\in\mathcal{J}}\left(C^{i^+}_{tj}(\xi_{[t]})i^+_{tj}(\xi_{[t]})+C^{i^-}_{tj}(\xi_{[t]})i^-_{tj}(\xi_{[t]})+C^{y}_{tj}(\xi_{[t]})y_{tj}(\xi_{[t]})\right)+C^{o}_{t}(\xi_{[t]})o_t(\xi_{[t]})\right)\right]$$

- **Constraints 1. State equations**: For all $t \in \mathcal{T}, j \in \mathcal{J}$, w.p.1,

$$i^-_{tj}(\xi_{[t]}) - i^+_{tj}(\xi_{[t]}) + i^+_{t-1,j}(\xi_{[t-1]}) - i^-_{t-1,j}(\xi_{[t-1]}) + x_{t-1,j}(\xi_{[t]}) = D_{tj}(\xi_{[t]})$$

- **Constraints 2. Overtime measurements**: For all $t \in \mathcal{T}$, w.p.1,

$$\sum_{j\in\mathcal{J}}(TS_j y_{tj}(\xi_{[t]}) + TB_j x_{tj}(\xi_{[t]})) - o_t(\xi_{[t]}) \leq C_t$$

- **Constraints 3. Link setup and production decisions**: For all $t \in \mathcal{T}, j \in \mathcal{J}$, w.p.1,

$$M_{tj}\, y_{tj}(\xi_{[t]}) \geq x_{tj}(\xi_{[t]})$$

- **Constraints 4. Capacity bounds**: For all $t \in \mathcal{T}, j \in \mathcal{J}$, w.p.1,

$$\begin{aligned} &i^+_{tj}(\xi_{[t]}) \leq I_{tj} \\ &i^+_{tj}(\xi_{[t]}) + x_{tj}(\xi_{[t]}) \leq I_{t+1,j} \\ &o_t(\xi_{[t]}) \leq O_t \end{aligned}$$

- **Constraints 5. Nature of variables**: For all $t \in \mathcal{T}, j \in \mathcal{J}$, w.p.1,

$$\begin{aligned} &x_{tj}(\xi_{[t]}), i^+_{tj}(\xi_{[t]}), i^-_{tj}(\xi_{[t]}), o_t(\xi_{[t]}) \geq 0 \\ &y_{tj}(\xi_{[t]}) \in \{0,1\} \end{aligned}$$

Note that in this optimization model, we assume that there is no production related cost. More detailed explanations of the model can be found in (Daryalal et al. 2020).

6.2. Data description

Regarding the stochastic process $\{D_{tj}(\xi_{[t]})\}_{t\in\mathcal{T},j\in\mathcal{J}}$, for demand scenario generation purposes, we used two publicly available datasets: Walmart Sales Forecasting data (Kaggle 2014) and Corporación Favorita Grocery Sales Forecasting data (Kaggle 2018). For the remaining optimization model parameter generation, we followed the steps described in (Daryalal et al. 2020) which builds on the steps provided in (Helber et al. 2013). We next briefly outline the characteristics of the used demand data.

The Walmart dataset contains 12 columns of weekly sales data records for each department at each store, from 2010-02-05 to 2012-11-01. We let each combination of "*(store, department)*" be one demand time-series. Given the dataset, we decided to drop seven columns due to a large amount of missing data and a low correlation with the other features. We note that removal of these features would not lead to any predictive performance degradation, and producing robust and reliable statistical

Table 1. Notation used in the MSlag model

Sets and Indices:			
$\mathcal{T}$	Time periods, $t \in \mathcal{T}$		
$\mathcal{J}$	Product types, $j \in \mathcal{J}$		
Parameters:			
Deterministic:		*Uncertain:*	
M_{tj}:	Big-M value for modeling	$\xi_{[t]}$	Vector of all random variables up to period t
TS_j	Setup time	$D_{tj}(\xi_{[t]})$	Demand of products
TB_j:	Unit production time	$C^y_{tj}(\xi_{[t]})$	Production cost
C_t:	Production capacity	$C^{i^-}_{tj}(\xi_{[t]})$	Backlog cost
I_{tj}:	Inventory capacity	$C^{i^+}_{tj}(\xi_{[t]})$	Inventory cost
O_t:	Overtime bound	$C^o_t(\xi_{[t]})$	Overtime cost
Decision variables:			
$x_{tj}(\xi_{[t]})$	Production level		
$i^+_{tj}(\xi_{[t]})$	Inventory level		
$i^-_{tj}(\xi_{[t]})$	Backlog level		
$o_t(\xi_{[t]})$	Overtime measurement		
$y_{tj}(\xi_{[t]})$	Production decision which takes value 1 if production is setup, 0 otherwise		

forecasting results using missing data is beyond the scope of this paper. We directly used and/or modified the remaining features to obtain new covariates to be used as an input for DeepAR: Year, Month, Week, Day, Temperature, Fuel price, and Holiday indicator. Despite the fact that temperature and fuel price might not be the most suitable covariates as they include data on future events, for this research, we assumed both time series as given, thanks to the advances in the field of time-series forecasting. For the sake of prediction, we extracted the data of the last eight weeks and trained all prediction models for the data without the last eight weeks. For (store, department) combinations to be used in our numerical evaluation, we randomly sampled from a subset of the original dataset where the forecasting methods had a noticeable predictive performance difference.

For the Walmart and Favorita datasets, the planning horizons correspond to the time steps $\{135, \ldots, 142\}$ and $\{580, \ldots, 591\}$, as can be observed in Figure 4 and Figure 8, respectively. We use all the data available before the beginning of the planning horizon in the training phase. In the rest of paper, we use τ to denote the *prediction horizon length* for DeepAR.

6.3. Implementation details

All the MSP model policy generation and evaluation algorithms are implemented in Python, and Gurobi 9.0.0. is used as a mixed-integer programming solver. Experiments are conducted on a MacOS workstation with Quad-Core 3GHz Intel i5-8500B CPU and 16 GB memory.

Regarding DeepAR, a PyTorch (Paszke et al. 2019) (a deep learning package for Python) implementation developed by Zhang et al. (2019) was used as the baseline

code. We note that there exists an open source package called GluonTS (Alexandrov et al. 2020), which is capable of performing deep learning-based time-series forecasting in an integrated and automated manner. However, to the best of our knowledge, the conditional sample path generation feature is not supported by GluonTS.

7. Numerical results

In this section, we first compare the predictive performance of the three forecasting methods. In the policy evaluation section, we analyze the impact of improved forecasts on the deterministic and two-stage policies. For the former, in addition to the risk-neutral setting, we consider a risk-averse approach where the demand values are replaced by the highest value of their corresponding prediction interval.

7.1. Forecasting performance

We evaluate the overall predictive performance of AR(1), moving average and DeepAR models on the used datasets. Specifically, for the Walmart dataset and the Favorita dataset, we use regular moving average (MA) and logaritmic moving average (LogMA) models, where we consider the moving average parameters of 8 and 12, respectively, both obtained via grid search. Our choice of LogMA rather than MA for the Favorita dataset is based on the fact that it has been empirically shown to be competitive with the state-of-the-art methods (Kaggle 2018). In terms of the effort required to build the forecasting models, the parameters associated with ARIMA-based models are calculated in less than ten minutes, while training DeepAR is significantly more time consuming, requiring around twelve hours.

In order to train the DeepAR model, we split the dataset into three subsets: training, validation, and test. For the Walmart dataset, training, validation, and test datasets involve time steps $\{0, \ldots, 126\}$, $\{127, \ldots, 134\}$, and $\{135, \ldots, 142\}$, respectively. Similarly, for the Favorita dataset, training, validation, and test datasets involve time steps $\{0, \ldots, 529\}$, $\{530, \ldots, 579\}$, and $\{580, \ldots, 591\}$, respectively. We performed hyper-parameter tuning for the model parameters, where the model is trained with the training dataset, and its performance is measured based on the validation dataset. The obtained DeepAR hyper-parameters are provided in Table 2. Using the identified parameters, we performed time series forecasting for the time steps corresponding to the test dataset.

Table 2. DeepAR hyper-parameters

	Batch size	**Learning rate**	**LSTM layers**	**LSTM units**
Favorita	64	1e-3	3	40
Walmart	64	1e-3	6	80

We use a rolling-horizon framework to evaluate different policies, and thus adopt conditional sampling. In rolling-horizon framework, at every decision-making stage, that is, for any MSP model to be solved, the forecasts are obtained using available history, i.e., all the true observations made up to that stage. For the first stage (i.e., for the first MSP to be solved), forecasts are called *unconditioned* (or *0-step conditioned*); for the second one they are referred to as *1-step conditioned*, and so on. For t'-step conditioned case, we refer to t' as the *conditioning step*. In order to provide a better idea

on the notion of conditioning, we illustrate unconditioned and one-step conditioned probabilistic forecasts of AR(1) and DeepAR for one particular time series from the Walmart dataset in Figure 4.

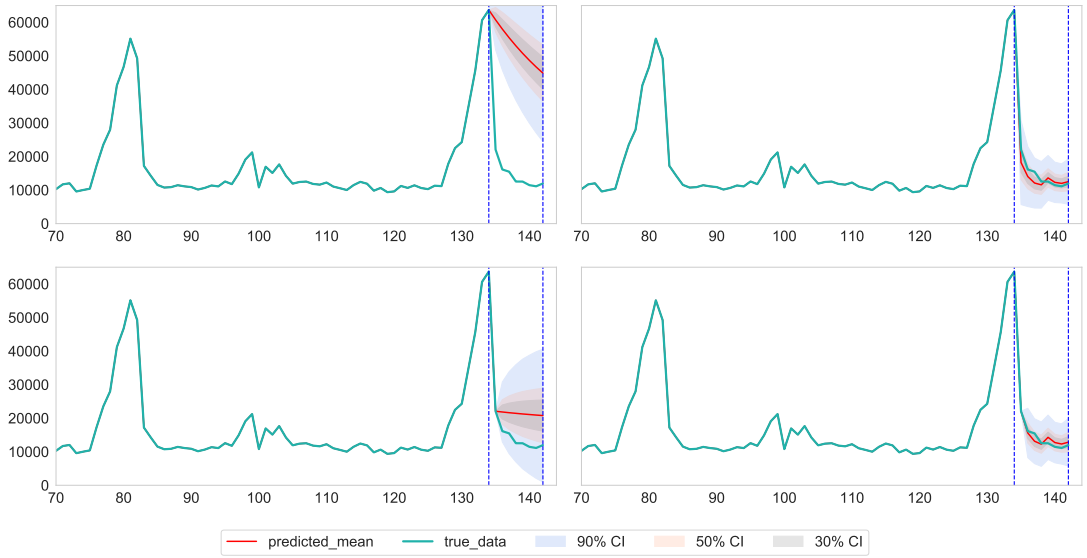


Figure 4. Probabilistic product demand forecasting for a selected time-series from the Walmart data (top-left: unconditioned AR(1), bottom-left: 1-step conditioned AR(1), top-right: unconditioned DeepAR, bottom-right: 1-step conditioned DeepAR). The first decision-making stage, $t = 0$, corresponds to the time step 135. Dotted vertical blue lines indicate the beginning and the end of the planning horizon for MSlag.

Figure 4 demonstrates the poor-quality performance of AR(1) compared to that of DeepAR, for both unconditioned and 1-step conditioned forecasts, in terms of both the predicted mean values and confidence interval widths. Although we provide the forecasting results for only one particular product in this figure as an example, we note that the plots for the remaining products look very similar. In what follows, we present a formal overall performance measure and compare all three considered methods.

Since DeepAR is used as one of our main forecasting methods, we rely on the so-called normalized deviation (ND) performance metric (Salinas et al. 2019). However, as predictions are performed for different step conditions in our rolling-horizon framework, namely from unconditioned to $(T-1)$-step conditioned if the planning horizon length is T, we define and use ND values for each possible conditioning step. More specifically, given T as the length of the planning horizon, and a conditioning step $t' \in \{0, \ldots, T-1\}$, the *$t'$-step ND value* is computed as

$$\mathrm{ND}(t', T) := \frac{\sum_{t=t'+1}^{T} \sum_{j \in \mathcal{J}} |D_{tj} - \hat{D}_{tj}|}{\sum_{t=t'+1}^{T} \sum_{j \in \mathcal{J}} |D_{tj}|}$$

where for (the time-series corresponding to) product j at time t, D_{tj} and $\hat{D}_{tj}$ denote the true and predicted median demand values.

We report the obtained ND values of selected set of time series, for each conditioning time step for the Walmart and Favorita datasets in Table 3, where column labels refer to the conditioning steps (t').

Table 3. ND(t', T) values at each prediction period for different forecasting methods

(a) Walmart dataset, $T = 8$, $\tau = 8$

Method / t'	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
AR(1)	0.34	0.36	0.28	0.18	0.16	0.15	0.17	0.14
MA	0.25	0.25	0.19	0.14	0.13	0.12	0.13	0.13
DeepAR	0.17	0.14	0.13	0.12	0.13	0.12	0.14	0.15

(b) Favorita dataset, $T = 12$, $\tau = 12$

Method / t'	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
AR(1)	0.99	1.22	0.56	0.56	0.60	0.60	0.56	0.55	0.50	0.41	0.37	0.31
LogMA	0.50	0.47	0.44	0.43	0.44	0.43	0.42	0.42	0.41	0.37	0.35	0.31
DeepAR	0.48	0.46	0.43	0.39	0.40	0.40	0.41	0.42	0.43	0.41	0.37	0.45

Table 3a shows that for the Walmart dataset DeepAR performs the best for the first four time steps, while it is competitive with the other methods for the last four time steps, namely $t' \in \{4, 5, 6, 7\}$. In contrast, AR(1) performs the worst for the Walmart dataset for most of the time steps. For the Favorita dataset, as seen in Table 3b, while DeepAR outperforms the others for the majority of the time steps, the performance gap between LogMA and DeepAR is not as large as in the Walmart case, which is inline with the reported performance of LogMA in (Kaggle 2018). Note that the forecasting performance of all the methods for stages closer to the end of the planning horizon, i.e., the ones obtained by using a large conditioning step value, are highly accurate. This can be attributed to the fact that the value of the target time-series stabilizes as time index increases, while there is some unstable behavior early on, e.g., a huge peak is observed at the beginning of the horizon (see Figure 4). It is also important to recognize that time-series forecasting methods usually perform well for short-term prediction, while they frequently fail to provide high-accuracy long-term predictions.

As the ND values are computed based on predicted median demand values, they provide insights mostly on the quality of point forecasts, which play an important role in risk-neutral deterministic policies. On the other hand, for two-stage policies, as well as for risk-averse deterministic policies, we instead use probabilistic forecasts. That is, we rely on the provided confidence intervals to either sample scenarios or estimate the worst-case scenario/build an uncertainty set. Therefore, we use an additional metric to be able to assess the overall performance of the considered methods in terms of their probabilistic forecasts. In that regard, we rely on the ρ-risk (quantile loss) metric (Salinas et al. 2019), which is used to quantify the forecasting accuracy of a quantile ρ of the predictive probability distribution.

Given $\rho \in (0, 1)$, the ρ-risk of a product $j \in \mathcal{J}$ is defined as

$$L_\rho(Z_j, \hat{Z}_j^\rho) = 2 \cdot (\hat{Z}_j^\rho - Z_j) \cdot \left(\rho \cdot \mathbb{I}_{(\hat{Z}_j^\rho > Z_j)} - (1 - \rho) \cdot \mathbb{I}_{(\hat{Z}_j^\rho \leq Z_j)}\right)$$

where $\mathbb{I}_{(.)}$ is the indicator function. $Z_j = \sum_{t_0+L}^{t_0+L+S} D_{tj}$ where t_0 is the forecast start

point, L is the lead time after the forecast start point, and S is the length of the prediction range. In our case, t_0 corresponds to the start of the planning horizon (i.e., the period index of our $t = 0$), there is no lead time, i.e., $L = 0$, and the length of the prediction range is equal to the length of the planning horizon, i.e., $S = T$. Lastly, the predicted ρ-quantile value of Z_j is denoted by $\hat{Z}_j^\rho$. Note that having a lower value of ρ-risk is better. Analogously, ρ-risk for the entire set of products, ρ-risk, is computed by using the following formula:

$$\frac{\sum_{j \in \mathcal{J}} L_\rho(Z_j, \hat{Z}_j^\rho)}{\sum_{j \in \mathcal{J}} Z_j}$$

For our datasets, using a high confidence level of $\rho = 0.9$, for the Walmart dataset, we obtain the ρ-risk values of 1.57, 1.57 and 1.08 for AR(1), MA and DeepAR, respectively. On the other hand, for the Favorita dataset, the 0.9-risk values are attained as 2.32, 1.65 and 1.27 respectively for AR(1), MA and DeepAR. These values indicate that DeepAR is able to generate more accurate probabilistic forecasts with lower variance.

Lastly, we note that we are not aware of any previous studies reporting the forecasting performance values (e.g. ρ-risk) of the Walmart and Favorita datasets. However, the above-mentioned values seem to be inline with the ones provided in Table 1 of (Salinas et al. 2019). For a challenging natural number valued dataset, namely PARTS, the authors report 0.9-risk value as 1.06 when $L = 0$ and $S = 8$ (as in our Walmart dataset). Also, for another dataset, called ELECTRICITY, which can be considered as a favorable dataset for forecasting due to inherent seasonality, they find the 0.9-risk as 1.33 when $L = 3$ and $S = 12$ (as in our Favorita set).

7.2. Policy evaluation

In this section, we present the quality of different policies combined with different forecasting methodologies. More specifically, we focus on deterministic and two-stage policies (see Section 4.1 and Section 4.2) as well as a risk-averse (robust) policy optimizing the worst-case cost instead of the expected cost.

As a key performance measure we consider the percentage gap between the perfect information (PI) bound and the objective function value obtained by evaluating considered policies in a rolling-horizon framework. We refer to this value as "Gap %". The PI bound can be obtained by solving the optimization problem described in Section 4.1, by replacing ξ_t^{expected} with $\hat{\xi}_t$ (i.e., replacing the expected value with the true observation) for every stage $t \in \mathcal{T}$. The choice of the PI bound as a reference point is common in the MSP literature, and stems from the facts that the optimal solution to an MSP is not possible to obtain in general, that the PI bound is computationally quite cheap to compute, and that finding improved bounds are usually computationally demanding. In our experiments, we consider multiple product groups, where a group corresponds to a particular set $\mathcal{J}$ in MSlag, as such we report the Gap % values averaged over all the product groups.

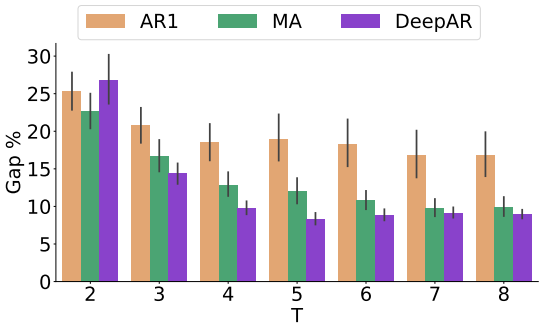
We note that the MSP model has relatively complete recourse. As such inaccurate forecasts do not lead to any infeasibility. However, in such cases, backlog penalties are incurred, which increases the model objective function values, implying larger Gap % values.

7.2.1. Deterministic risk-neutral setting

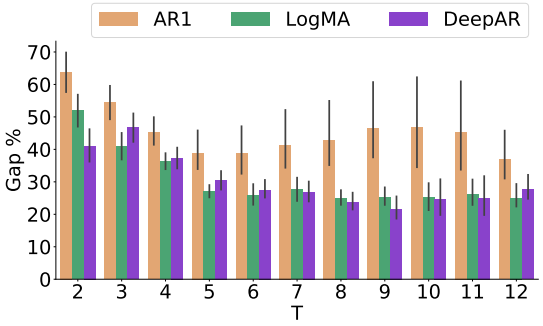
In the rolling-horizon experiments with the deterministic policy, we consider 50 product groups (i.e., $\mathcal{J}$ sets) and use the following parameter settings:

- The Walmart dataset: $|\mathcal{J}| = 10$ and $T \in \{2, 3, \ldots, 8\}$
- The Favorita dataset: $|\mathcal{J}| = 15$ and $T \in \{2, 3, \ldots, 12\}$

We provide the results obtained with AR(1), (Log)MA, and DeepAR forecasting methods for varying length of the planning horizon in Figure 5, which also includes error bars for each case, showing the deviation in different product groups.



(a) the Walmart dataset

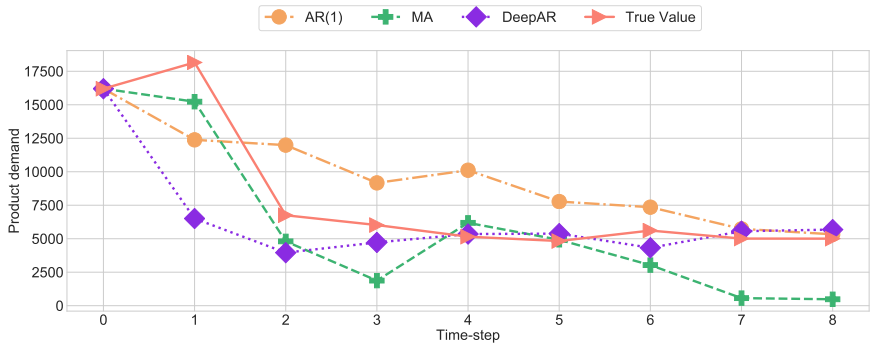


(b) the Favorita dataset

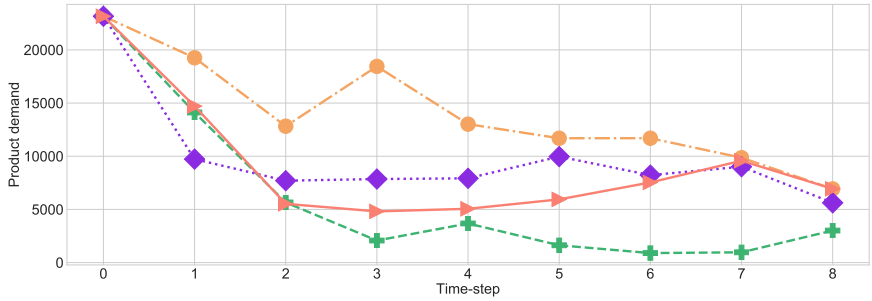
Figure 5. Gap % values for different planning horizons (T) under risk-neutral setting

We observe that, for the deterministic policy, overall, DeepAR leads to the lowest Gap % values, while AR(1) consistently performs the worst. We also note that the benefits of improved forecasting can be substantial. For instance, for the Walmart dataset with $T = 8$, the average Gap % values for AR(1), MA and DeepAR are 16.8%, 10.0% and 9.0%, respectively. Furthermore, for the same settings (Walmart with $T = 8$), average of the worst and best three product group Gap % values for AR(1) are 46.6% and 7.0%, whereas, DeepAR leads to 12.3% and 6.5% gaps for the same product groups, which shows that a significantly higher-quality deterministic policies can be obtained with the help of improved forecasts. For both datasets, we find that MA-based methods and DeepAR perform similarly, which is consistent with their predictive performance reported in Section 7.1. In addition, these two methods show low variability in Gap % values, and, in general, their average performance as well as the variance improve as T gets closer to the prediction horizon length, τ. We note that our analysis with varying number of products ($|\mathcal{J}|$) for fixed T values did not yield substantially different insights, thus was omitted for the sake of brevity.

We observe a poor performance of DeepAR for the Walmart dataset when $T = 2$. To elaborate, we provide Figure 6 which illustrates the forecasting result comparison of a sample path generated from each forecasting method, for two different time-series. In



(a)



(b)

Figure 6. Forecasting results for two different time-series from the Walmart dataset.

Figure 6a, after the large peak at $t = 0$ (corresponding to time index 135 in the original time-series) as shown in Figure 4, DeepAR predictions tend to rapidly decrease with respect to the true values, producing some errors in the first few steps of the forecasting horizon. On the other hand, after the first few steps, DeepAR produces significantly better results compared to AR(1) and MA. These forecasting results help explaining why DeepAR performs worse when $T = 2$, while performing well overall. We also note that a similar phenomenon is observed in Figure 6b and in many other time-series, where in this case, DeepAR failed to perform better than MA.

We also provide the average performance values for the Walmart dataset when $T = 2$ in Table 4, which points to the inferior performance of DeepAR for $t' \in \{0, 1\}$. Similar to Table 3, column indices of Table 4 correspond to the time step where the forecasting method was conditioned.

Table 4. ND(t', T) values for the Walmart dataset when $T = 2$.

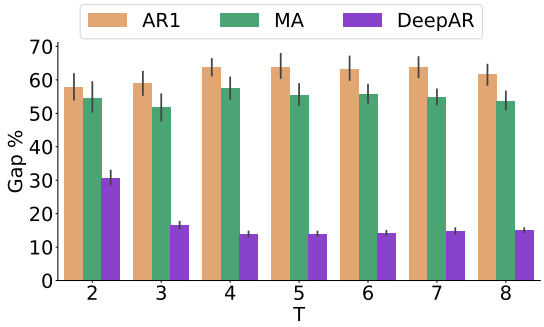
Method	$t' = 0$	$t' = 1$
AR(1)	0.22	0.23
MA	0.20	0.22
DeepAR	0.26	0.19

In the Appendix, we provide the ND(t', T) values for all T and $t' < T$ combinations, of the DeepAR model used in our analysis (i.e., the one trained with τ equal to the largest T option) for both datasets. We note that for $t' = 0$, the only case where DeepAR performance was worse than the other methods was for the Walmart case with $T = 2$.

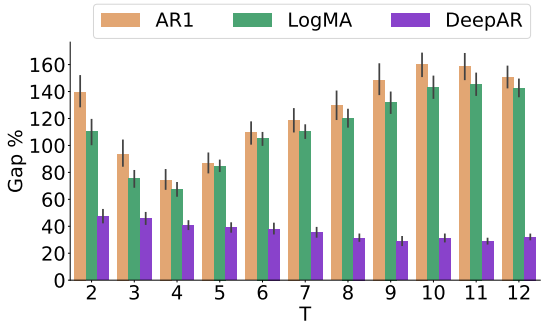
7.2.2. Deterministic risk-averse setting

In this section, we present the deterministic policy evaluation results under a risk-averse setting. Specifically, for each forecasting method, we generate a 90% prediction interval for the predictions of each time series, and use its maximum value to be taken as the value of the demand in the worst case. These worst-case demand estimates are provided to the MSlag models to be solved in the rolling-horizon framework.

Figure 7 shows the obtained results with the risk-averse setting for varying planning horizon lengths. We observe that the policies generated with DeepAR significantly



(a) the Walmart dataset



(b) the Favorita dataset

Figure 7. Gap % values for different planning horizons (T) under risk-averse setting

outperform the others. In particular, for the Favorita dataset, while LogMA-based deterministic policies are comparable to the DeepAR-based ones in the risk-neutral setting, LogMA loses its strength in the risk-averse setting. For instance, for the Favorita dataset with $T = 12$, the average Gap % values are 150.7%, 142.7%, and 32.7% for AR(1), LogMA, and DeepAR models, respectively.

As discussed by Salinas et al. (2019), DeepAR tends to yield tighter prediction intervals. Such a phenomenon can also be observed from, for instance, Figure 8 which presents sample forecasting results from the Favorita dataset, showing the significant reduced-variance forecasting power of DeepAR. We note that the ρ-risk measure analysis provided in Section 7.1 supports this finding. Accordingly, the worst-case predic-

tions of DeepAR are typically closer to the ground truth, which considerably benefits deterministic robust policies. In this regard, we note that considering a sophisticated forecasting method can be highly beneficial not only in terms of uncertainty modeling for stochastic programming, but also to construct smaller and accurate uncertainty sets for robust optimization models.

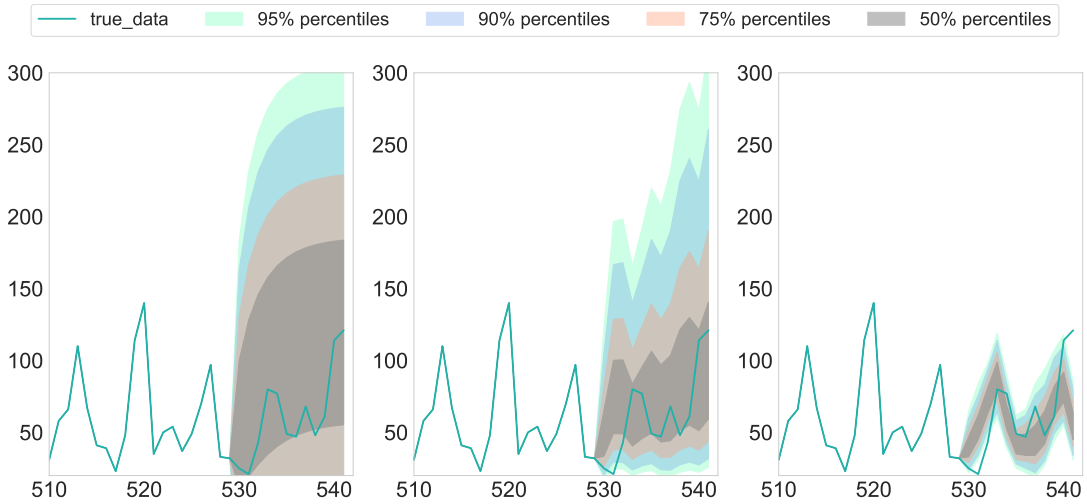


Figure 8. A graphical representation of forecasting results of AR(1), LogMA and DeepAR for the first portion of validation dataset of the Favorita dataset, from left to right respectively, where its x-axis is time steps and y-axis is product demand

Lastly, regarding the results presented in Figure 5 and Figure 7, note that the risk-averse policies typically yield larger Gap % values than the risk-neutral ones since they are more conservative. However, in our experiments with DeepAR, they turn out to be quite close to each other. For example, for the Walmart dataset with $T = 8$, Gap % values are 9.0% and 15.0% for risk-neutral and risk-averse settings, respectively, while for the Favorita dataset with $T = 12$, these values are 27.7% and 32.1%.

7.2.3. Two-stage setting

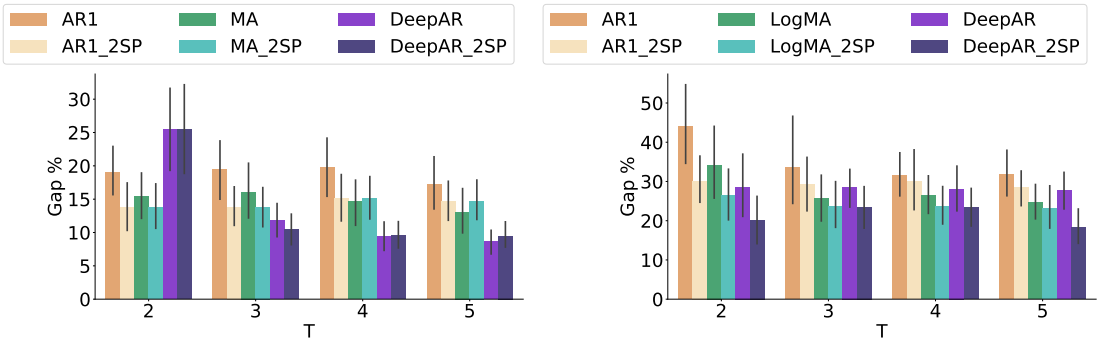
Finally, we present our findings on the impact of improved forecasts for two-stage policies using a rolling-horizon approach. In these experiments, we consider the number of scenarios, $|\mathcal{S}|$, as 9 and 15 for the Walmart and Favorita datasets, respectively. Unlike the deterministic policy experiments, for both datasets, we consider 30 product groups with $|\mathcal{J}| = 3$ and a range of planning horizon lengths $T \in \{2, 3, 4, 5\}$. Such a restricted setup consideration is mainly because two-stage stochastic models to be solved in the rolling horizon framework are significantly more computationally demanding than the deterministic models. We note that as we have mixed-binary recourse decisions, we solve the two-stage stochastic programs via their extensive form; developing a problem-specific solution method (e.g., decomposition algorithm) is out of scope of the paper.

We chose the number of scenarios (9 and 15 for the two datasets) via preliminary experiments. We tested different number of sample sizes by enforcing a time limit of three hours. The provided sample sizes are the maximum ones that could be handled given the time limit, which is for all the product groups and the full rolling-horizon framework, rather than for solving a single two-stage stochastic program. More specifically, in our rolling-horizon experiments for each of the 30 product groups, we simulate

a sample path by solving $T-1$ two-stage stochastic programs. As such, although we can solve a single two-stage stochastic program with more than 9 and 15 scenarios in the Walmart and Favorita cases, respectively, considering the total computational effort of solving $30(T-1)$ stochastic programs, we opted for a smaller sample size. It is also worth to mention that the low-variance forecasts provided by DeepAR would reduce the need for a large number of scenarios to get high-quality solutions for the two-stage stochastic programs.

A particularly useful feature of DeepAR for our purposes is that it can generate sample paths with all non-negative demand values by appropriately adjusting the distribution assumption of the neural network architecture (e.g., using negative binomial or Poisson distribution for natural number time-series predictions). This is not the case in ARMA models, since the white noise term is not bounded. Therefore, as product demand is assumed to be non-negative valued, for sample paths obtained by the ARMA-based models, we replace negative prediction values with zeros.

Figure 9 provides results on two-stage policies as well as their deterministic counterparts for all the forecasting methods and varying planning horizon lengths. First, we observe that, overall, the two-stage policies obtained by using the scenarios sampled from the DeepAR (DeepAR_2SP), perform the best. Note that the Walmart dataset with $T = 2$ is an exception as discussed in our analysis with the deterministic policies and will be omitted in what follows.



(a) Walmart dataset

(b) Favorita dataset

Figure 9. Gap % values for deterministic and two-stage policies in the risk-neutral setting

In particular, for the Favorita dataset, two-stage policies consistently provide better results compared to their deterministic counterparts for all the forecasting methods. Most notably, DeepAR_2SP substantially improves over its deterministic counterpart, and outperforms all the other approaches for this dataset.

Second, there are various cases where the deterministic policies outperform the stochastic policies. For instance, in the Walmart dataset, DeepAR deterministic policies provide better Gap % values than ARMA-based two-stage policies. This finding confirms that, well-suited forecasting models for a problem can lead to high-quality solutions via computationally cheap heuristics, while sophisticated optimization models with low-accuracy forecasts are bound to provide low-quality solutions. Considering the fact that AR(1) model is frequently used in the MSP literature, our study suggests to first assess its predictive performance on the considered application in order not to sacrifice the chance of obtaining high-quality policies.

One important caveat in our assessment is that our evaluations are solely based

on a single out-of-sample scenario, namely, the one consisting of the true realizations of all the random demand variables. However, in quantifying the value of stochastic solutions, the expected value is used as the measure, that is, the average performance over a (large) set of potential realizations of random variables is taken as the reference, since the ground truth is not known. On the contrary, in our analysis, we intentionally spared some portion of the historical data and evaluated the performance of alternative policies based on the true realizations in order to provide a better picture for decision makers as only one particular scenario is realized in real life. Therefore, although two-stage policies are expected to perform better than the deterministic policies on the average, in our reported results, there are various cases that we observe the opposite as in the case of DeepAR for the Walmart dataset with $T = \{4, 5\}$.

While our analysis with (Log)MA forecasting method for the risk-neutral deterministic policies provide competitive results to those of DeepAR, the gap between the two widens for two-stage policies. As in the deterministic risk-averse case, this can be attributed to the fact that DeepAR provides substantially better probabilistic forecasting results with tight confidence intervals (e.g. see Figure 4, Figure 8 and ρ-risk values provided in Section 7.1), which are used to sample more informative scenarios with a high confidence level.

Lastly, we note that while we considered a risk-averse approach for the deterministic setting using risk-averse forecasts, we have not considered a risk-averse approach for the two-stage setting. A suitable option for the latter can be obtaining a policy based on two-stage robust optimization models, which would be computationally difficult to solve even for small instances in the presence of mixed-integer recourse.

8. Conclusion

In this paper, we analyze the impact of deep learning-based time-series forecasting on MSP model outcomes. In our analysis, we consider two commonly adopted MSP policies (deterministic, and two-stage stochastic), and compare three different forecasting methods, namely, AR(1), (Log)MA, and DeepAR. Our numerical study with a practical MSP application provides evidence that improved forecasts can lead to significant benefits in terms of the performance of the look-ahead policies. Specifically, we observe that, with an improved forecasting method, deterministic heuristics have potential to provide high-quality policies. Moreover, once the forecasting quality is assured, the use of stochastic programming can further improve the overall quality of the results. We note that these findings align well with those of (Gauvin et al. 2018a,b) who emphasize the importance of better forecasting methods for improved policies. Lastly, our results suggest that DeepAR can be especially useful as a sampling tool for stochastic programming, as well as to construct uncertainty sets or estimate the worst-case scenario for robust optimization, due to its impressive probabilistic forecasting capabilities.

Considering the success of state-of-the-art deep learning-based probabilistic forecasting methods, as well as their highly suitable features for MSPs, we believe that the proposed approach of combining them with MSP policy generators has significant potential to be utilized widely in practice. As this would allow generating high-quality policies via simpler heuristic, thus substantially reduce the computational burden, it can widen the applicability of multistage stochastic programming. We note that in particular for applications involving complex time series such as stock price prediction, integration of MSP with deep learning-based time series forecasting methods might provide significantly better results than that of with simple forecasting methods that

are not able to produce reliable predictions. However, for some other applications such as electricity generation planning where time series shows noticeable patterns such as seasonality and cyclic behaviour which can be accurately predicted with simple autoregressive models, combination of MSP with deep learning-based time-series forecasts might not produce a drastic improvement.

Our study constitutes the first step in leveraging the power of state-of-the-art probabilistic forecasting methods in stochastic optimization. There are many intriguing future research directions. First, our computational study can be extended through different MSP applications (e.g. hydro-thermal electricity generation) and other probabilistic forecasting methods (e.g. deep state space models (Rangapuram et al. 2018)), which would be helpful in further assessing the benefits of improved forecasts. Second, in our experiments with two-stage policies, where we used crude Monte Carlo sampling, alternative sampling approaches such as Quasi-Monte Carlo sampling (Homem-de Mello and Bayraksan 2014) and Monte Carlo Markov Chain based importance sampling (Parpas et al. 2015) can be employed to see how much variance reduction can be accomplished, and accordingly whether better two-stage policies can be obtained in a reduced computational effort (i.e., using a fewer number of scenarios). Third, it would be interesting to evaluate the impact of improved forecasts on MSP policies constructed via more sophisticated scenario tree-free methods (e.g., decision rule-based approaches) Lastly, the experimental framework that we considered belongs to the "traditional data-driven sample average approximation approach" (Kannan et al. 2020), as we first constructed the time series forecasting model, and then used the predictions to build the optimization model. In this regard, combining deep learning-based time series forecasting methodologies with other class of modern data-driven optimization frameworks such as "Predict, then Optimize" (Elmachtoub and Grigas 2021) is an intriguing future research direction.

Acknowledgment

This research is supported by a grant from LG SciencePark. M. Bodur also acknowledges the support of the Natural Sciences and Engineering Research Council of Canada.

Data availability statement

The data that supports the findings of this study are openly available at:
https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting
https://www.kaggle.com/c/favorita-grocery-sales-forecasting

References

R. K. Agrawal, F. Muchahary, and M. M. Tripathi. Long term load forecasting with hourly predictions based on long-short-term-memory networks. In *2018 IEEE Texas Power and Energy Conference (TPEC)*, pages 1–6. IEEE, 2018.

A. Alexandrov, K. Benidis, M. Bohlke-Schneider, V. Flunkert, J. Gasthaus, T. Januschowski, D. C. Maddix, S. Rangapuram, D. Salinas, J. Schulz, L. Stella, A. C. Türkmen, and Y. Wang. GluonTS: Probabilistic and Neural Time Series Modeling in Python. *J. Mach. Learn. Res.*, 21(116):1–6, 2020. http://jmlr.org/papers/v21/19-820.html.

A. Alonso-Ayuso, L. F. Escudero, and M. T. Ortuno. BFC, a branch-and-fix coordination algorithmic framework for solving some types of stochastic pure and mixed 0–1 programs. *European J. Oper. Res.*, 151(3):503–519, 2003.

I. Bakir, N. Boland, B. Dandurand, and A. Erera. Sampling scenario set partition dual bounds for multistage stochastic programs. *INFORMS J. Comput.*, 32(1):145–163, 2020.

D. Bampou and D. Kuhn. Scenario-free stochastic programming with polynomial decision rules. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 7806–7812. IEEE, 2011.

D. Bertsimas and A. Georghiou. Design of near optimal decision rules in multistage adaptive mixed-integer optimization. *Oper. Res.*, 63(3):610–627, 2015.

J. R. Birge. Aggregation bounds in stochastic linear programming. *Math. Program.*, 31(1): 25–41, 1985a.

J. R. Birge. Decomposition and partitioning methods for multistage stochastic linear programs. *Oper. Res.*, 33(5):989–1007, 1985b.

A. Bischi, L. Taccari, E. Martelli, E. Amaldi, G. Manzolini, P. Silva, S. Campanari, and E. Macchi. A rolling-horizon optimization algorithm for the long term operational scheduling of cogeneration systems. *Energy*, 184:73–90, 2019.

M. Bodur and J. R. Luedtke. Two-stage linear decision rules for multi-stage stochastic programming. *Math. Program.*, pages 1–34, 2018.

G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

C. CarøE and R. Schultz. Dual decomposition in stochastic integer programming. *Oper. Res. Lett.*, 24(1):37–45, 1999.

D.-S. Chen, R. G. Batson, and Y. Dang. Applied integer programming. *Hoboken, NJ*, 2010.

X. Chen, M. Sim, P. Sun, and J. Zhang. A linear decision-based approximation approach to stochastic programming. *Oper. Res.*, 56(2):344–357, 2008.

Y. Cheng, C. Xu, D. Mashima, V. L. Thing, and Y. Wu. Powerlstm: Power demand forecasting using long short-term memory neural network. In *International Conference on Advanced Data Mining and Applications*, pages 727–740. Springer, 2017.

M. Daryalal, M. Bodur, and J. R. Luedtke. Lagrangian dual decision rules for multistage stochastic mixed integer programming. *arXiv preprint arXiv:2001.00761*, 2020.

O. Dowson and L. Kapelevich. SDDP.jl: A julia package for stochastic dual dynamic programming. *INFORMS J. Comput.*, 2020.

A. N. Elmachtoub and P. Grigas. Smart "predict, then optimize". *Manag. Sci.*, 2021.

H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. Deep learning for time series classification: A review. *Data Min. Knowl. Discov.*, 33(4):917–963, 2019.

J. C. B. Gamboa. Deep learning for time-series analysis. *arXiv preprint arXiv:1701.01887*, 2017.

C. Gauvin, E. Delage, and M. Gendreau. A stochastic program with time series and affine decision rules for the reservoir management problem. *European J. Oper. Res.*, 267(2):716–732, 2018a.

C. Gauvin, E. Delage, and M. Gendreau. A successive linear programming algorithm with non-linear time series for the reservoir management problem. *Comput. Manag. Sci.*, 15(1): 55–86, 2018b.

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

S. Helber, F. Sahling, and K. Schimmelpfeng. Dynamic capacitated lot sizing with random demand and dynamic safety stocks. *OR spectrum*, 35(1):75–105, 2013.

T. Homem-de Mello and G. Bayraksan. Monte carlo sampling-based methods for stochastic optimization. *Surv. Oper. Res. Manag. Sci.*, 19(1):56–85, 2014.

Kaggle. *Walmart Recruiting - Store Sales Forecasting*, 2014. https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting.

Kaggle. *Corporación Favorita Grocery Sales Forecasting, Can you accurately predict sales for a large grocery chain?*, 2018. https://www.kaggle.com/c/favorita-grocery-sales-forecasting/.

R. Kannan, G. Bayraksan, and J. R. Luedtke. Data-driven sample average approximation with covariate information, 2020. http://www.optimization-online.org/DB_FILE/2020/07/7932.pdf.

L. Kuan, Z. Yan, W. Xin, C. Yan, P. Xiangkun, S. Wenxue, J. Zhe, Z. Yong, X. Nan, and Z. Xin. Short-term electricity load forecasting method based on multilayered self-normalizing gru network. In *2017 IEEE Conference on Energy Internet and Energy System Integration (EI2)*, pages 1–5. IEEE, 2017.

Á. Lorca, M. Favereau, and D. Olivares. Challenges in the management of hydroelectric generation in power system operations. *Current Sustainable/Renewable Energy Reports*, 7: 94–99, 2020.

G. Lulli and S. Sen. A branch-and-price algorithm for multistage stochastic integer programming with application to stochastic batch-sizing problems. *Manag. Sci.*, 50(6):786–796, 2004.

P. Parpas, B. Ustun, M. Webster, and Q. K. Tran. Importance sampling in stochastic programming: A Markov chain Monte Carlo approach. *INFORMS J. Comput.*, 27(2):358–377, 2015.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

M. V. Pereira and L. M. Pinto. Multi-stage stochastic optimization applied to energy planning. *Math. Program.*, 52(1):359–375, 1991.

S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski. Deep state space models for time series forecasting. In *Advances in neural information processing systems*, pages 7785–7794, 2018.

D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *Int. J. Forecast.*, 2019.

A. Shapiro. Analysis of stochastic dual dynamic programming method. *European J. Oper. Res.*, 209(1):63–72, 2011.

A. Shapiro and A. Nemirovski. On complexity of stochastic programming problems. In *Continuous optimization*, pages 111–146. Springer, 2005.

A. Shapiro, W. Tekaya, J. Paulo da Costa, and M. Pereira Soares. Risk neutral and risk averse stochastic dual dynamic programming method. *European J. Oper. Res.*, 224(2):375 – 391, 2013.

A. Shapiro, D. Dentcheva, and A. Ruszczynski. *Lectures on stochastic programming: modeling and theory*. SIAM, 2014.

J. Silvente, G. M. Kopanos, V. Dua, and L. G. Papageorgiou. A rolling horizon approach for optimal management of microgrids under stochastic uncertainty. *Chemical Engineering Research and Design*, 131:293–317, 2018.

S. W. Wallace and W. T. Ziemba. *Applications of stochastic programming*, volume 5. SIAM, 2005.

J.-P. Watson and D. L. Woodruff. Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. *Comput. Manag. Sci.*, 8(4):355–370, 2011.

H. Ye, J. Luedtke, and H. Shen. Call center arrivals: When to jointly forecast multiple streams? *Prod. Oper. Manag.*, 28(1):27–42, 2019.

Y. Zhang, Q. Jiang, and X. Ma. A Pytorch implementation for DeepAR, 2019. https://github.com/zhykoties/TimeSeries.

J. Zou, S. Ahmed, and X. A. Sun. Stochastic dual dynamic integer programming. *Math. Program.*, 175(1):461–502, 2019.

Appendix A. ND values for the Walmart dataset

In this section, we provide $\mathrm{ND}(t', T)$ values for the Walmart dataset when different planning horizon lengths, T values, are assumed. Note that all the ND values are calculated using the same models.

- $T = 2$

Method / t'	**0**	**1**
AR(1)	0.22	0.23
MA	0.20	0.22
DeepAR	0.26	0.19

- $T = 3$

Method / t'	**0**	**1**	**2**
AR(1)	0.27	0.31	0.26
MA	0.24	0.27	0.22
DeepAR	0.23	0.17	0.15

- $T = 4$

Method / t'	**0**	**1**	**2**	**3**
AR(1)	0.30	0.34	0.27	0.13
MA	0.25	0.27	0.20	0.12
DeepAR	0.20	0.15	0.13	0.11

- $T = 5$

Method / t'	**0**	**1**	**2**	**3**	**4**
AR(1)	0.32	0.36	0.28	0.16	0.14
MA	0.26	0.27	0.20	0.13	0.12
DeepAR	0.19	0.15	0.13	0.12	0.13

- $T = 6$

Method / t'	**0**	**1**	**2**	**3**	**4**	**5**
AR(1)	0.32	0.35	0.26	0.16	0.12	0.07
MA	0.25	0.25	0.19	0.13	0.12	0.09
DeepAR	0.18	0.14	0.12	0.11	0.11	0.09

- $T = 7$

Method / t'	**0**	**1**	**2**	**3**	**4**	**5**	**6**
AR(1)	0.33	0.36	0.27	0.17	0.14	0.12	0.14
MA	0.25	0.25	0.19	0.13	0.12	0.10	0.11
DeepAR	0.17	0.14	0.12	0.12	0.12	0.11	0.13

Appendix B. ND values for the Favorita dataset

In this section, we provide $\mathrm{ND}(t', T)$ values for the Favorita dataset when different planning horizon lengths, T values, are assumed. Note that all the ND values are calculated using the same models.

- $T = 2$

Method / t'	**0**	**1**
AR(1)	0.86	1.58
LogMA	0.68	0.60
DeepAR	0.59	0.55

- $T = 3$

Method / t'	**0**	**1**	**2**
AR(1)	0.95	1.35	0.47
LogMA	0.61	0.51	0.42
DeepAR	0.61	0.60	0.59

- $T = 4$

Method / t'	**0**	**1**	**2**	**3**
AR(1)	0.89	1.14	0.43	0.37
LogMA	0.54	0.46	0.40	0.35
DeepAR	0.53	0.49	0.44	0.30

- $T = 5$

Method / t'	**0**	**1**	**2**	**3**	**4**
AR(1)	0.89	1.11	0.44	0.40	0.42
LogMA	0.52	0.45	0.39	0.36	0.36
DeepAR	0.51	0.47	0.43	0.34	0.38

Method / t'	0	1	2	3	4	5
AR(1)	0.93	1.16	0.49	0.46	0.53	0.61
LogMA	0.51	0.46	0.42	0.40	0.42	0.42
DeepAR	0.49	0.46	0.42	0.34	0.37	0.35

- $T = 6$
- $T = 7$

Method / t'	0	1	2	3	4	5	6
AR(1)	0.97	1.20	0.52	0.51	0.58	0.63	0.47
LogMA	0.51	0.47	0.42	0.40	0.43	0.41	0.37
DeepAR	0.49	0.46	0.42	0.35	0.37	0.36	0.36

- $T = 8$

Method / t'	0	1	2	3	4	5	6	7
AR(1)	1.00	1.25	0.55	0.55	0.63	0.67	0.57	0.61
LogMA	0.52	0.47	0.43	0.42	0.44	0.44	0.41	0.42
DeepAR	0.48	0.45	0.41	0.35	0.38	0.37	0.36	0.36

- $T = 9$

Method / t'	0	1	2	3	4	5	6	7	8
AR(1)	1.06	1.31	0.60	0.61	0.70	0.75	0.70	0.76	0.78
LogMA	0.53	0.49	0.45	0.44	0.47	0.47	0.46	0.49	0.53
DeepAR	0.48	0.46	0.42	0.36	0.39	0.39	0.39	0.41	0.46

- $T = 10$

Method / t'	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
AR(1)	1.06	1.32	0.61	0.62	0.70	0.73	0.68	0.72	0.67	0.48
LogMA	0.53	0.50	0.46	0.45	0.47	0.48	0.47	0.49	0.49	0.42
DeepAR	0.50	0.47	0.44	0.39	0.41	0.42	0.43	0.46	0.51	0.52

- $T = 11$

Method / t'	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
AR(1)	1.03	1.27	0.59	0.59	0.64	0.66	0.61	0.62	0.55	0.43	0.37
LogMA	0.52	0.48	0.45	0.44	0.45	0.44	0.43	0.43	0.42	0.37	0.33
DeepAR	0.48	0.46	0.43	0.38	0.40	0.40	0.40	0.41	0.43	0.40	0.28