FUSING OPTIMIZATION PROXIES AND REINFORCEMENT LEARNING FOR MULTISTAGE STOCHASTIC OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Multistage stochastic programs provide a general framework for sequential decision-making under uncertainty, but their complexity grows exponentially with the number of decision epochs. A popular policy for multistage stochastic optimization solves a two-stage stochastic program at every time step in a rolling-horizon manner, relaxing non-anticipatory constraints for subsequent stages. Solving this sequence of optimization models is computationally expensive, limiting real-time deployment for some classes of applications. This paper presents PROSE (Proxies Reinforcement-learned in Optimization-Structured Environments), which fuses optimization proxies with deep reinforcement learning to obtain a feasible policy for the multistage problem. The optimization model defines the learning problem: at each epoch, the policy takes the same inputs as the two-stage program, including its weighted scenarios. Its action space is the feasible set of the optimization model, enforced by a differentiable repair layer applied to a neural prediction, and its reward is the first-stage cost. The policy is trained by backpropagating the cumulative realized cost through repair and state transitions over complete episodes, end-to-end in a self-supervised manner. On energy dispatch and inventory replenishment case studies, PROSE implements a feasible action at every decision, is 1,600$\times$ faster per decision, and typically incurs a lower cumulative realized cost than the rolling two-stage policy.

1 INTRODUCTION

In many operational settings, sequential decisions under uncertainty constrain the ones that follow: generator ramp limits restrict successive dispatch adjustments, and inventory carried forward shapes later fulfillment options. Each action may depend only on the information available when it is implemented, which makes these problems multistage. An exact formulation represents uncertainty as a scenario tree, in which every node branches into the outcomes possible at the next decision epoch and carries its own decisions. Even two outcomes per epoch yield 2^T scenarios after T epochs; with hundreds of epochs in an operating day, the problem cannot be tractably formulated, let alone solved between consecutive decisions.

The rolling-horizon two-stage approximation remains a popular policy in practice. At each time step, it selects an action across a finite set of scenarios and gives each scenario its own recourse plan over a finite horizon. However, this approach has two limitations. First, constructing and solving a new stochastic program at every step is expensive and limits real-time use. Second, letting recourse adapt to each scenario relaxes non-anticipativity after the current epoch, and the resulting anticipativity error persists with more scenarios. The policy, meanwhile, incurs the cost accumulated along the realized trajectory, which the two-stage program's per-epoch objective does not measure.

Each limitation can be addressed by a specific line of research. Optimization proxies replace each solve with a neural mapping from the instance to an action and make decisions orders of magnitude faster, but they are trained on individual instances and therefore reproduce the per-epoch approximation. Deep reinforcement learning trains a policy on the cost it accumulates, but it usually handles hard constraints through penalties, so its actions may not be feasible. In the domains considered in this paper, feasibility is essential: an infeasible action cannot be executed safely.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

This paper develops PROSE (Proxies Reinforcement-learned in Optimization-Structured Environments), which fuses optimization proxies with deep reinforcement learning to learn a feasible policy for the multistage problem. At each step, the optimization model defines the learning problem. The policy observes the instance of the two-stage program: the current state, the observations revealed at this stage, and weighted scenarios of the future. Its action space is the feasible set of the optimization model: a neural network proposes an action, and a state-dependent repair maps the proposal into that set. Its "reward" is the first-stage cost, i.e., the cost of implementing the action, and its state evolves through the system's state transition. The policy is trained by backpropagating the cumulative realized cost through repair and state transitions over complete episodes in a self-supervised manner. At inference, each decision is one forward pass followed by repair. The approach assumes continuous decisions and complete recourse: the first assumption ensures that these gradients are well-defined, and the second assumption ensures that a feasible action at time t always leaves a feasible action at time $t + 1$.

The contributions of PROSE are as follows:

- To our knowledge, PROSE is the first policy for multistage stochastic optimization that fuses optimization proxies with deep reinforcement learning in a self-supervised manner: the optimization model supplies the observation, action space, reward, and dynamics, and the policy is trained on the closed-loop cost of its own rollouts.
- PROSE features a state-dependent, piecewise-differentiable repair that ensures the feasibility of every action. Its gradients propagate the rollout cost through repair and state transitions, allowing the policy to be trained end to end.
- Computational results show that the PROSE policy is 1,638–2,145$\times$ faster per decision and incurs a lower cumulative realized cost than the rolling two-stage policy, which relaxes non-anticipatory constraints and does not benefit from end-to-end training.

2 Related Work

Two-stage stochastic look-ahead and learned recourse. At each decision epoch, online anticipatory algorithms replace a multistage stochastic problem with a two-stage relaxation, in which the current decision is shared across sampled scenarios and multiperiod recourse is planned separately for each, and reoptimize as information arrives. Mercier & Van Hentenryck (2007) split the resulting loss into a sampling error that decays as scenarios are added and an anticipativity error that does not, bounded by the problem's anticipatory gap. Neur2SP speeds up each solve by embedding learned recourse values in a first-stage optimization model (Dumouchelle et al., 2022). In contrast, PROSE solves no optimization model online.

Optimization proxies and reinforcement learning. E2ELR combines neural prediction with closed-form feasibility repair for single-period problems (Chen et al., 2024), and Ye et al. (2026) train a proxy for sequential stochastic programs to imitate solutions at states recorded along solver trajectories. Model-free deep reinforcement learning trains policies from sampled returns (Schulman et al., 2017) and usually imposes constraints in expectation through penalties or Lagrangian terms (Achiam et al., 2017), so individual actions can violate hard operating limits. RLOP combines the two: it trains a proxy to imitate a model predictive controller (MPC) and then refines it with score-function policy gradients on closed-loop reward (Yuan et al., 2022). In contrast, PROSE is trained from scratch on the realized cost of its own trajectories, with pathwise gradients and no solver labels, and keeps every implemented action feasible through repair.

Differentiable policies and feasibility layers. When the dynamics are known and differentiable, policies can be trained on trajectory cost with pathwise gradients: HDPO backpropagates total cost through trajectories replayed on historical data (Alvo et al., 2025), and differentiable predictive control trains neural controllers with constraint penalties (Drgoňa et al., 2022b;a). Several methods place an optimization solve inside the policy to enforce feasibility: a quadratic-program projection at every step (Helm et al., 2026), projection onto state-dependent feasible sets (Chen et al., 2021), and optimization subproblems in the forward pass of two-stage deep decision rules (Rosemberg et al., 2025). In contrast, PROSE observes the scenarios of the rolling two-stage program at every epoch

and restores feasibility without a solver, onto a feasible set that changes with the state its own earlier actions produced.

3 Multistage Stochastic Program and Its Two-Stage Approximation

3.1 Multistage stochastic program

Consider an episode of T decision epochs, $t = 0, 1, \ldots, T-1$. At epoch t, the exogenous information h_t^{exo} comprises current observations, observation history, and external updates such as revised forecasts; the sequence $h_0^{\text{exo}}, \ldots, h_{T-1}^{\text{exo}}$ has joint distribution $\mathbb{P}$. The state z_t is endogenous: it depends on the actions implemented before t and on the information observed by then. The parameters η_t are the exogenous problem data known at epoch t, such as cost coefficients and parameters of interperiod coupling. Together, z_t and η_t define the current cost $C_t(x_t; z_t, \eta_t)$ and the feasible set of the current action,

$$\mathcal{X}_t(z_t, \eta_t) = \{x \in \mathbb{R}^{n_x} : g_t(x; z_t, \eta_t) \le 0, \quad h_t(x; z_t, \eta_t) = 0\}, \tag{1}$$

where the vector-valued functions g_t and h_t collect current inequality and equality constraints. After x_t is implemented, the state evolves according to

$$z_{t+1} = F_t(z_t, x_t, h_{t+1}^{\text{exo}}), \tag{2}$$

where F_t is the state transition. A policy π specifies, at each epoch t, a decision rule $x_t = \pi_t(z_t, h_t^{\text{exo}})$ that reads only the current state and the information observed through epoch t, so that no decision depends on uncertainty revealed after it is implemented. The multistage problem is

$$\begin{aligned} \min_{\pi} \quad & \mathbb{E}_{\mathbb{P}}\Big[\sum_{t=0}^{T-1} C_t(x_t; z_t, \eta_t)\Big] \\ \text{s.t.} \quad & x_t = \pi_t(z_t, h_t^{\text{exo}}) \in \mathcal{X}_t(z_t, \eta_t), \quad \forall\, t = 0, \ldots, T-1, \\ & z_{t+1} = F_t(z_t, x_t, h_{t+1}^{\text{exo}}), \qquad\quad \forall\, t = 0, \ldots, T-2, \end{aligned} \tag{3}$$

with z_0 given. Problem 3 is a finite-horizon Markov decision process with state (z_t, h_t^{exo}), action x_t, and cost C_t. On a scenario tree with b outcomes per epoch, each rule π_t becomes one action for each of the b^t histories at epoch t, so the problem grows exponentially with T.

3.2 Rolling two-stage approximation

The rolling two-stage approximation replaces Equation 3 at each epoch with a program over a finite planning horizon of H periods: the current action x_t is shared across scenarios, each scenario receives its own recourse trajectory over the $H-1$ future periods, and only x_t is implemented. Let Ω_t denote the set of all future paths over periods $t+1, \ldots, t+H-1$ that the model admits at epoch t, and let $\xi_t \in \Omega_t$ be the random future path, with distribution $\mathbb{P}_t$ conditional on (z_t, h_t^{exo}). Within the planning horizon, $\tau \in \{1, \ldots, H-1\}$ is the look-ahead offset, corresponding to future time $t+\tau$. The two-stage problem is

$$V_t = \min_{x_t \in \mathcal{X}_t(z_t, \eta_t)} \left\{ C_t(x_t; z_t, \eta_t) + \mathbb{E}_t[Q_t(x_t, \xi_t; z_t, \eta_t)] \right\}, \tag{4}$$

where $\mathbb{E}_t$ denotes the expectation under $\mathbb{P}_t$. For a specified path ξ, let $y = (y_1, \ldots, y_{H-1})$ collect future decisions. The recourse value is

$$\begin{aligned} Q_t(x, \xi; z_t, \eta_t) = \min_{y} \quad & \sum_{\tau=1}^{H-1} C_{t,\tau}^{\text{rec}}(y_\tau; x, z_t, \eta_t, \xi) \\ \text{s.t.} \quad & y_\tau \in \mathcal{Y}_{t,\tau}(z_t, \eta_t, \xi), \qquad\quad \tau = 1, \ldots, H-1, \\ & b_{t,1}(x, y_1; z_t, \eta_t, \xi) \le 0, \\ & b_{t,\tau}(y_{\tau-1}, y_\tau; z_t, \eta_t, \xi) \le 0, \qquad \tau = 2, \ldots, H-1. \end{aligned} \tag{5}$$

The set $\mathcal{Y}_{t,\tau} \subseteq \mathbb{R}^{n_{y,\tau}}$ includes the future variable restrictions and constraints local to period τ. The functions $b_{t,\tau}$ encode coupling between successive periods, including the link from x to the first future decision y_1. Any equality constraints in $b_{t,1}$ or $b_{t,\tau}$ ($\tau \ge 2$) can be expressed as paired inequalities. $C_{t,\tau}^{\text{rec}}$ is the recourse cost incurred at future time $t+\tau$. Q_t is the minimum total recourse cost over the look-ahead horizon.

Complete recourse. This paper assumes complete recourse. For every epoch t, every feasible action $x \in \mathcal{X}_t(z_t, \eta_t)$, and every path $\xi \in \Omega_t$, the recourse problem in Equation 5 is feasible and has a finite optimal value. Moreover, for $t < T-1$, the state $z_{t+1} = F_t(z_t, x, h_{t+1}^{\text{exo}})$ reached by implementing x admits a feasible action, $\mathcal{X}_{t+1}(z_{t+1}, \eta_{t+1}) \neq \emptyset$, for every exogenous information h_{t+1}^{exo} that can follow. A policy that enforces the current constraints at every epoch therefore remains feasible along its trajectory. Both case studies satisfy this assumption.

3.3 Finite-scenario representation

In computation, the expectation in Equation 4 is approximated with a finite scenario set $\Xi_t = \{(\xi_{t,s}, \rho_{t,s})\}_{s \in \mathcal{S}_t}$ of S_t paths in Ω_t, indexed by $s \in \mathcal{S}_t = \{1, \dots, S_t\}$, drawn from $\mathbb{P}_t$ with weights $\rho_{t,s} \geq 0$ that sum to one, and regenerated at every epoch. The known data and scenarios are assembled from the exogenous information, $(\eta_t, \Xi_t) = \mathcal{U}_t(h_t^{\text{exo}})$. To make first-stage nonanticipativity explicit, this paper introduces a scenario copy $x_{t,s}$ of the current action x_t for each $s \in \mathcal{S}_t$. Let $\mathcal{Y}_t(x, \xi; z_t, \eta_t)$ denote the joint feasible set of recourse decisions satisfying all constraints in Equation 5. The resulting approximation is

$$\begin{aligned} \widehat{V}_t = \min_{\substack{x_t, \\ \{x_{t,s}, y_{t,s}\}_{s \in \mathcal{S}_t}}} \quad & C_t(x_t; z_t, \eta_t) + \sum_{s \in \mathcal{S}_t} \rho_{t,s} \sum_{\tau=1}^{H-1} C_{t,\tau}^{\text{rec}}(y_{t,s,\tau}; x_{t,s}, z_t, \eta_t, \xi_{t,s}) \\ \text{s.t.} \quad & x_t \in \mathcal{X}_t(z_t, \eta_t), \\ & y_{t,s} \in \mathcal{Y}_t(x_{t,s}, \xi_{t,s}; z_t, \eta_t), \qquad s \in \mathcal{S}_t, \\ & x_{t,s} = x_t, \qquad s \in \mathcal{S}_t \quad \text{(non-anticipativity)}. \end{aligned} \tag{6}$$

The equalities $x_{t,s} = x_t$ enforce nonanticipativity only for the current action: each scenario is an entire future path, and the recourse decisions $y_{t,s}$ are chosen as if that path were known in advance. For the rest of the paper, direct optimization means solving Equation 6, which maps its inputs to the implemented action, $x_t = \mathcal{M}_t(z_t, \eta_t, \Xi_t)$, where $\mathcal{M}_t$ denotes the optimization solver at epoch t. Figure 1 shows this cycle at two consecutive epochs. Repeating it defines an implementable policy for Equation 3, but not an optimal one, because this scenario-specific recourse relaxes nonanticipativity after the current epoch.

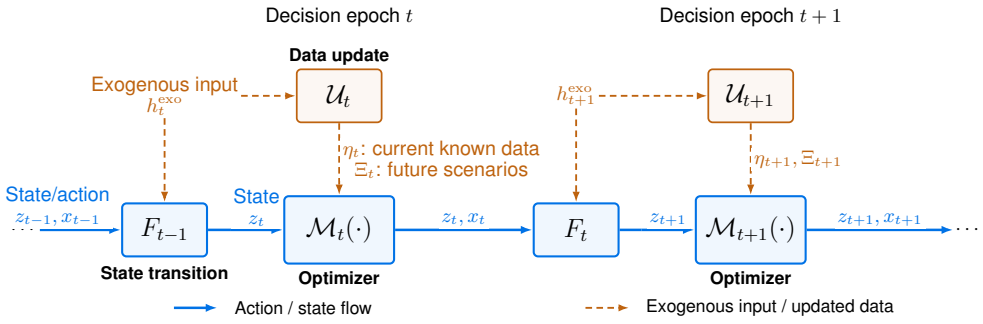

Figure 1: Two consecutive epochs of the rolling two-stage policy: $\mathcal{U}_t$ assembles the program's data, $\mathcal{M}_t$ solves for the implemented action x_t and passes z_t unchanged, and F_t forms the next state.

4 The PROSE Method

4.1 Policy: prediction and repair

PROSE learns a policy for Problem 3 in which the optimization model supplies every element: the policy observes the instance $I_t = (z_t, \eta_t, \Xi_t)$ of the two-stage program, acts in its feasible set $\mathcal{X}_t(z_t, \eta_t)$, incurs its first-stage cost C_t, and moves between states via F_t. The policy itself is an optimization proxy: a neural network f_θ followed by a state-dependent repair $\mathcal{R}_\kappa$ into the feasible set,

$$u_t = f_\theta(I_t), \qquad \tilde{x}_t = \pi_{\theta,\kappa}(I_t) = \mathcal{R}_\kappa(u_t; I_t). \tag{7}$$

Here u_t is the raw neural proposal, in the same units as the action, and $\tilde{x}_t$ is the repaired action. The state z_t in I_t is the previous state carried forward by Equation 2 (Figure 2). The network f_θ may be any differentiable architecture; each case study in Section 5 specifies its own.

The subscript κ specifies which constraints of Equation 1 the repair enforces, defining the set $\mathcal{X}_t^\kappa(z_t, \eta_t)$ that contains $\tilde{x}_t$, so training and deployment may use different repairs. The deployment repair κ_{dep} enforces all of them, so every implemented action is feasible. The training repair κ_{tr} may enforce a subset and penalize the rest in the training loss (Section 4.2), which lets training use a simpler repair while deployment remains feasible. All reported results use κ_{dep}.

Assumptions. Training differentiates the rollout cost with respect to θ (Section 4.3), so actions are assumed continuous and the training repair, F_t, and C_t are assumed to be piecewise differentiable. The repairs in both case studies satisfy this because they enforce bounds and linear totals whose limits depend on the state. Clipping to bounds is piecewise linear. Proportional rescaling moves the proposal toward its bounds until a total is met; once the direction of the move is fixed, it is linear in the limits and a smooth ratio in the proposal (Appendix B).

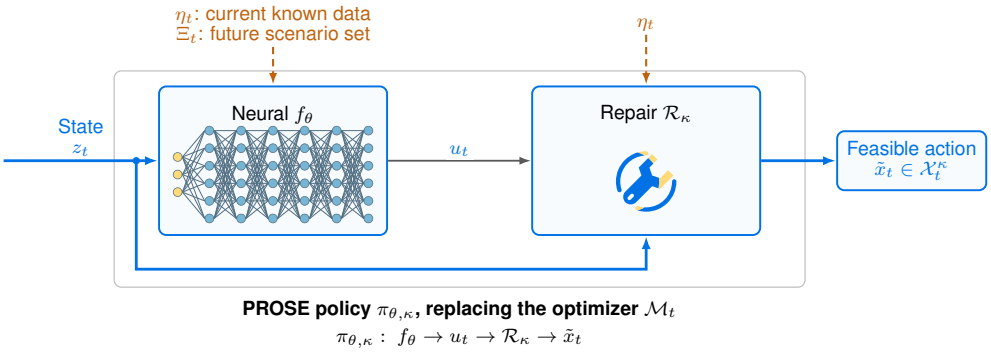

Figure 2: PROSE at one epoch, in place of the solver $\mathcal{M}_t$ of Figure 1: the network f_θ proposes u_t, and the state-dependent repair $\mathcal{R}_\kappa$ returns a feasible action $\tilde{x}_t$ without an optimization solve.

4.2 Label-free rollout objective

Training rolls the policy out over complete episodes. Episode d starts from an initial state $z_{d,0}$, independent of θ. At each epoch, the exogenous information $h_{d,t}^{\text{exo}}$ forms the instance $I_{d,t}$ through $(\eta_{d,t}, \Xi_{d,t}) = \mathcal{U}_t(h_{d,t}^{\text{exo}})$, the policy returns $\tilde{x}_{d,t}$ by Equation 7, and Equation 2 carries it to the next state $z_{d,t+1}$. Episodes are grouped into batches $\mathcal{B}$, within which θ is held fixed and gradients are accumulated before each parameter update.

For episode d over decision epochs $t = 0, \ldots, T-1$, the cumulative cost of the policy $\pi_{\theta,\kappa}$ is

$$J_d(\pi_{\theta,\kappa}) = \sum_{t=0}^{T-1} C_t(\tilde{x}_{d,t}; z_{d,t}, \eta_{d,t}), \tag{8}$$

where C_t is the first-stage cost of Equation 4 at the implemented action. Since $I_{d,t}$ depends only on the state and the information through epoch t, $\pi_{\theta,\kappa}$ is a policy for Problem 3, and minimizing the average of J_d over training episodes is empirical policy optimization for that problem: the scenarios shape each action through $I_{d,t}$, and the costs of later epochs price its consequences. Constraints that the training repair leaves unenforced are penalized by

$$P_\kappa(x; I_{d,t}) = \alpha_\kappa^\top \max\{g_t(x; z_{d,t}, \eta_{d,t}), 0\} + \beta_\kappa^\top |h_t(x; z_{d,t}, \eta_{d,t})|,$$

with g_t, h_t from Equation 1, componentwise maximum and absolute value, and nonnegative coefficients $\alpha_\kappa, \beta_\kappa$ that are zero for enforced constraints. A movement term discourages large repair corrections, giving the per-step training loss at epoch t

$$\ell_\kappa(\tilde{x}_{d,t}, u_{d,t}; I_{d,t}) = C_t(\tilde{x}_{d,t}; z_{d,t}, \eta_{d,t}) + P_\kappa(\tilde{x}_{d,t}; I_{d,t}) + \lambda_{\text{move}} \|\tilde{x}_{d,t} - u_{d,t}\|_2^2. \tag{9}$$

where $\lambda_{\text{move}} \geq 0$ weights the movement term. The training objective averages the per-step loss over a batch of size $|\mathcal{B}|$:

$$\mathcal{L}_{\mathcal{B}}(\theta; \kappa) = \frac{1}{|\mathcal{B}|T} \sum_{d \in \mathcal{B}} \sum_{t=0}^{T-1} \ell_{\kappa}(\tilde{x}_{d,t}, u_{d,t}; I_{d,t}), \tag{10}$$

which is evaluated along the rollout defined above, with $u_{d,t} = f_{\theta}(I_{d,t})$ and $\kappa = \kappa_{\text{tr}}$.

4.3 Pathwise gradients through the rollout

For a single training episode, gradients of the rollout objective propagate through the policy and state transitions, with the exogenous inputs and the training repair κ_{tr} held fixed.

A later cost depends on an earlier action through the state, along the chain

$$\theta \;\to\; u_t \;\to\; \tilde{x}_t \;\to\; z_{t+1} \;\to\; u_{t+1} \;\to\; \tilde{x}_{t+1} \;\to\; \ell_{t+1}.$$

Differentiating along this path gives the contribution

$$\frac{\partial \ell_{t+1}}{\partial \tilde{x}_{t+1}} \frac{\partial \tilde{x}_{t+1}}{\partial z_{t+1}} \frac{\partial z_{t+1}}{\partial \tilde{x}_t} \frac{\partial \tilde{x}_t}{\partial \theta}, \qquad \frac{\partial \tilde{x}_t}{\partial \theta} = \frac{\partial \mathcal{R}_{\kappa}}{\partial u_t} \frac{\partial f_{\theta}}{\partial \theta}, \tag{11}$$

where $\tilde{x}_t$ and $\tilde{x}_{t+1}$ are the policy outputs of Equation 7 at their respective epochs. The final derivative, $\partial \tilde{x}_t / \partial \theta$, differentiates through both the neural inference f_{θ} and the repair $\mathcal{R}_{\kappa}$, on the branch of the repair that is active at u_t.

Because z_{t+1} is an input to both the network f_{θ} and the repair $\mathcal{R}_{\kappa}$, the chain rule gives

$$\frac{\partial \tilde{x}_{t+1}}{\partial z_{t+1}} = \frac{\partial \mathcal{R}_{\kappa}}{\partial u_{t+1}} \frac{\partial f_{\theta}}{\partial z_{t+1}} + \frac{\partial \mathcal{R}_{\kappa}}{\partial z_{t+1}}.$$

The same expansion continues back to $\tilde{x}_0$, and summing the contributions over epochs and episodes gives $\mathrm{d}\mathcal{L}_{\mathcal{B}} / \mathrm{d}\theta$; Appendix C.3 writes it out in full.

Pathwise versus log-derivative gradients. Model-free methods instead sample actions from a stochastic policy $\pi_{\phi}(\cdot \mid I_{d,t})$ with parameters ϕ and use the log-derivative (likelihood-ratio) estimator

$$\nabla_{\phi} \, \mathbb{E}\big[J_d(\pi_{\phi})\big] = \mathbb{E}\Big[J_d(\pi_{\phi}) \sum_{t=0}^{T-1} \nabla_{\phi} \log \pi_{\phi}(x_{d,t} \mid I_{d,t})\Big],$$

where $x_{d,t} \sim \pi_{\phi}(\cdot \mid I_{d,t})$ is the sampled action, J_d is the cumulative cost of Equation 8, and the expectation is over episodes and sampled actions. The estimator itself needs no labels, but its variance grows with the action dimension and the horizon, so training from a random initialization can be slow or fail to converge. RLOP, for example, first fits the policy to MPC solutions, which serve as labels, before refining it with policy gradients (Yuan et al., 2022). PROSE's policy is deterministic, and because the dynamics and repair are known and differentiable, its pathwise gradient along each training episode is computed analytically, which lets it be trained without labels.

4.4 Truncated and full unrolling

The gradient in Section 4.3 propagates every loss back through all earlier epochs of the episode; this is *full unrolling*. *Truncated unrolling* runs the same forward rollout but backpropagates each loss only to the start of its window of W consecutive decisions (Figure 3). This paper chooses W to divide T, and full unrolling is the case $W = T$. At each window boundary, the stop-gradient operator sg passes the state forward unchanged but sets its derivative to zero, so both regimes produce the same trajectory and losses and differ only in their gradients.

The two regimes trade bias against stability. Full unrolling keeps every dependency and yields the exact gradient of the episode cost, but the products in Equation 11 then span hundreds of decision epochs and can explode or vanish. Truncation bounds their length at the cost of dependencies longer than W decision epochs.

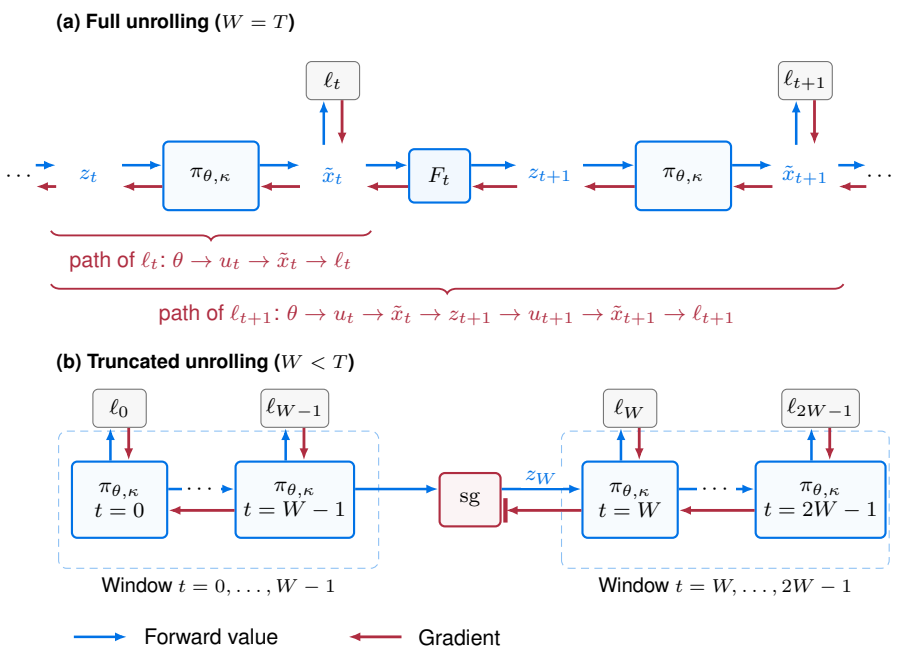

Figure 3: Temporal credit assignment. Braces mark how far back each per-step loss reaches. (a) Gradients pass through repair and the state transition into all earlier epochs. (b) At a window boundary, sg passes z_W forward but cuts the gradient.

5 Experiments

5.1 Evaluation protocol

PROSE and the rolling two-stage program are compared as decision policies with the same information at each epoch. Each policy starts from the same initial state and follows its own trajectory. The comparison reports the realized cost J_d of Equation 8 and the per-decision runtime, summarized by its mean and 95th percentile; every implemented action of every policy is feasible under an independent audit. Let $\nu_{d,t}^{\mathrm{2SP}}$ denote the runtime of solving the two-stage program and $\nu_{d,t}^{\mathrm{PROSE}}$ denote the runtime of PROSE at decision t of episode d. The speedup is reported as the median of the paired ratios: $\mathrm{median}_{d,t}\ \nu_{d,t}^{\mathrm{2SP}} / \nu_{d,t}^{\mathrm{PROSE}}$.

Each case study is split chronologically into training, validation, and test blocks (70/15/15%). In the energy dispatch case study, PROSE is trained with $W \in \{12, 24\}$ and with full unrolling ($W = 276$); in the inventory case study, with $W = 64$. Appendix C.2 details the splits, hyperparameter tuning, checkpoint selection, and test coverage.

5.2 Stochastic energy dispatch

Setup. This case study formulates a rolling two-stage program, adapted from the stochastic look-ahead dispatch model of Zhao et al. (2024) and detailed in Appendix A. Optimization instances are built on the RTS-GMLC system (Grid Modernization Lab Consortium, n.d.), a three-area, 73-bus power network with real-time profiles of demand, renewable availability, and reserve requirements (Appendix C.1). At each five-minute epoch, the action $x_t = (p_t, r_t)$ sets generator dispatch and reserves, the state z_t is the previous dispatch, which constrains p_t through ramp limits, and Ξ_t contains ten weighted demand and renewable paths over a twelve-period horizon ($S_t = 10$, $H = 12$). The baseline solves this program with line overload penalized as a monetary cost, which its reported cost includes.

The network f_θ is a multilayer perceptron with hidden widths 512, 256, and 128, ReLU activations, and dropout 0.1, applied to z_t, η_t, and the ten demand and renewable paths in Ξ_t. It emits one logit

	Operating cost		Per-decision runtime		
Policy	Total ($M) ↓	Gap (%) ↓	Mean (ms) ↓	p95 (ms) ↓	Speedup ↑
Two-stage program	99.333	—	27,460.04	34,172.74	1×
Truncated, $W = 12$	95.059	−4.30	**13.56**	**15.05**	**2,145**×
Truncated, $W = 24$	**94.980**	**−4.38**	17.79	18.19	1,638×
Full, $W = 276$	95.079	−4.28	17.58	17.85	1,653×

Table 1: Energy dispatch over the 55 test episodes (15,169 decisions). The best value in each column is in bold.

Policy	Minimum	Q1	Median	Q3	Maximum	Mean
(a) Daily operating cost ($M) ↓						
Two-stage program	1.4286	1.5524	1.6912	2.0384	2.4880	1.8060
Truncated, $W = 12$	1.2072	**1.3899**	1.6796	2.0241	2.4398	1.7284
Truncated, $W = 24$	1.1988	1.3939	**1.6769**	**2.0217**	2.4265	**1.7269**
Full, $W = 276$	**1.1891**	1.4034	1.6834	2.0250	**2.4185**	1.7287
(b) Daily cost gap vs. two-stage program (%) ↓						
Truncated, $W = 12$	−17.48	**−7.72**	−2.15	−0.91	2.40	−4.85
Truncated, $W = 24$	−17.66	−7.63	**−2.99**	**−1.28**	2.48	**−4.91**
Full, $W = 276$	**−18.29**	−7.56	−2.79	−1.21	**2.38**	−4.80

Table 2: Daily costs over the 55 test days: (a) per policy and (b) paired gaps, computed day by day. Q1 and Q3 are the 25th and 75th percentiles; the best value in each column is in bold.

per generator, which a ramp-aware sigmoid maps into that generator's admissible range to form u_t. $\mathcal{R}_\kappa$ reconstructs reserves from the repaired dispatch; line limits are penalized during training and enforced at deployment (Appendix B).

Cost. The $W = 12$, $W = 24$, and fully unrolled PROSE configurations cost 4.30%, 4.38%, and 4.28% less than the two-stage program in total (Table 1). Table 2 compares the policies' day by day through the paired gap $100\,(J_d^{\mathrm{PROSE}} - J_d^{\mathrm{2SP}})/J_d^{\mathrm{2SP}}$: all three configurations cost less than the program on 47 of the 55 days, with mean paired gaps between -4.80% and -4.91%.

Speedup. The speedup is $2{,}145\times$ for $W = 12$, $1{,}638\times$ for $W = 24$, and $1{,}653\times$ for full unrolling. The two-stage program needs about 27 seconds per decision on average, whereas the 95th-percentile runtime of every PROSE configuration is under 20 milliseconds.

5.3 Multi-echelon inventory replenishment

Setup. The second case study uses daily demand, prices, initial stock, and lead times for 200 stock-keeping units (SKUs), that is, distinct products, from the Multi-Agent Benchmark for Inventory Management (MABIM) (Yang et al., 2023). In its serial network, vendor → store 3 → store 2 → store 1 → customer, each of the three nodes has 20,000 units of storage shared by all SKUs, which couples their decisions. The state z_t contains inventory at each node, outstanding shipments by remaining lead time, and backlog; the action $x_t \in \mathbb{R}_+^{3\times 200}$ sets, for every SKU, the vendor order and the two internal transfers, 600 shipment quantities in total. After the action, demand is realized, sales are fulfilled from retailer stock, unfilled demand becomes backlog, and arrivals beyond a node's capacity are rejected as overflow.

The baseline is a two-stage program with the 600 current shipment quantities shared across ten equal-weight, 32-day historical demand paths and all later shipments, sales, inventory, backlog, and overflow as scenario-dependent recourse; each solve has 698,600 variables, and its look-ahead holds prices at their current values.

The network shares one set of weights across SKUs. For each SKU it combines current inventory, pipeline, backlog, and prices with lead-time demand summaries and the average of learned 128-dimensional encodings of the ten demand paths; a 256-dimensional local encoder with mean and maximum pooling across SKUs then feeds a shared head that emits the three edge logits. The

	Realized outcomes						Per-decision runtime		
Policy	Profit ($M) ↑	Gap (%) ↑	Fulfilled (%) ↑	Mean backlog ↓	Terminal backlog ↓	Over-flow ↓	Mean (ms) ↓	p95 (ms) ↓	Speedup ↑
Two-stage	18.441	—	99.388	**3,329.1**	4,112.0	2,778.5	14,969.88	34,575.46	1×
PROSE	**18.790**	**+1.89**	**99.607**	3,922.9	**2,640.1**	**0.0**	**4.96**	**5.30**	**1,990×**

Table 3: Inventory replenishment over the 177-day test trajectory, with the PROSE trained with $W = 64$. The best value in each column is in bold.

repair is a differentiable allocator that rations each destination's free storage across SKUs and limits transfers by source stock. It enforces every constraint in both training and deployment, so $P_\kappa = 0$, and overflow from later arrivals is priced as an operating cost. Because the same encoders and head are applied to every SKU and every demand path, and their outputs are combined by pooling and averaging, the model's 397,571 parameters do not grow with the number of SKUs or scenarios.

Performance is measured by the realized profit: sales revenue minus procurement, holding, backlog, and overflow costs, summed over one continuous 177-day test trajectory. Demand fulfillment, backlog, and overflow are also reported.

Profit and service. PROSE earns a profit of $18.790 million against $18.441 million for the two-stage program, an increase of $348,829 (1.89%), and fulfills 99.607% of demand against 99.388% (Table 3). Its mean backlog is 18% higher, but its terminal backlog is 36% lower and below its own mean: PROSE clears backlog as it goes, whereas the program's backlog builds up toward the end of the trajectory. Both profits include the backlog penalties as monetary costs. PROSE also incurs no overflow, whereas the program accepts 2,778.5 rejected arrival units as priced recourse.

Speedup. PROSE requires 4.96 ms per decision on average, compared with 14.97 s for the program, a 1,990× speedup; every timed solve of the program terminates optimally. PROSE's latency is also far more stable: its 95th-percentile runtime is 7% above its mean, whereas the program's is 2.3 times its mean.

5.4 Why PROSE outperforms

The two-stage program is optimal for a per-epoch objective that has three limitations relative to the closed-loop cost (Mercier & Van Hentenryck, 2007): its scenario-specific recourse plans each future as if it were known, which can misrank current actions; its horizon ends after H periods; and its finite scenario set adds sampling error. PROSE is trained on the closed-loop cost itself, with weights tuned over all epochs rather than overfitted to any single one, and it observes the same scenarios, which carry the information needed for good decisions when the anticipatory gap is small.

Energy dispatch. The three backward windows save nearly the same (Table 1), so longer credit assignment adds little, and the gap stems mainly from the first limitation: scenario-specific recourse misranking current actions. The saving concentrates on low-cost days (Table 2).

Inventory replenishment. Here the second limitation binds: long lead times push much of a vendor order's consequences beyond the look-ahead, whereas PROSE's loss prices them along its own trajectory. PROSE also learns to avoid overflow, which the program's scenario-specific, finite-horizon plans accept as priced recourse.

6 Conclusion

This paper fuses optimization proxies with deep reinforcement learning to learn feasible policies for multistage stochastic optimization: the optimization model supplies the observation, action space, reward, and dynamics, and PROSE is trained through repair and state transitions without optimizer labels or an embedded solver. In both case studies, PROSE is feasible at every decision, incurs lower cumulative cost than the rolling two-stage program, and is 1,638–2,145× faster per decision. Extending PROSE to discrete decisions is left to future work.

Reproducibility statement

Section 3 and Appendix A specify the optimization models, Section 4 and Appendix B specify the policy, repair, and training objective, and Appendix C gives the data preparation, splits, hyperparameters, and training configuration.

AI use statement

In this work, we used generative AI tools for literature review, brainstorming and refining research ideas, feedback on the methodology and experimental design, cleaning and preprocessing data, implementing methods, interpreting results, and drafting and polishing the text. We have not used generative AI tools to generate synthetic data sets, formulate mathematical claims, write proofs, or translate, and qualitative data analysis is not applicable to this work. The authors reviewed all AI-assisted content and verified generated code for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

References

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 22–31, 2017.

Matias Alvo, Daniel Russo, Yash Kanoria, and Minuk Lee. Deep reinforcement learning for inventory networks: Toward reliable policy optimization, 2025. URL https://arxiv.org/abs/2306.11246v3. Version 3; first posted in 2023.

Bingqing Chen, Priya Donti, Kyri Baker, J. Zico Kolter, and Mario Bergés. Enforcing policy feasibility constraints through differentiable projection for energy optimization. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*, 2021. doi: 10.1145/3447555.3464874. URL https://arxiv.org/abs/2105.08881v1.

Wenbo Chen, Mathieu Tanneau, and Pascal Van Hentenryck. End-to-end feasible optimization proxies for large-scale economic dispatch. *IEEE Transactions on Power Systems*, 39(2):4723–4734, 2024. doi: 10.1109/TPWRS.2023.3317352. URL https://arxiv.org/abs/2304.11726v2.

Ján Drgoňa, Sayak Mukherjee, Aaron Tuor, Mahantesh Halappanavar, and Draguna Vrabie. Learning stochastic parametric differentiable predictive control policies. *IFAC-PapersOnLine*, 55(25):121–126, 2022a. doi: 10.1016/j.ifacol.2022.09.334. URL https://arxiv.org/abs/2203.01447v2.

Ján Drgoňa, Aaron Tuor, and Draguna Vrabie. Learning constrained adaptive differentiable predictive control policies with guarantees, 2022b. URL https://arxiv.org/abs/2004.11184v6. Version 6; first posted in 2020.

Justin Dumouchelle, Rahul Patel, Elias B. Khalil, and Merve Bodur. Neur2SP: Neural two-stage stochastic programming. In *Advances in Neural Information Processing Systems*, volume 35, pp. 23992–24005, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9793671e4be9858a69a32545204d59d1-Paper-Conference.pdf.

Grid Modernization Lab Consortium. RTS-GMLC: Reliability test system data. GitHub repository, n.d. URL https://github.com/GridMod/RTS-GMLC/tree/master/RTS_Data. Data provided by DOE/NREL/Alliance for Sustainable Energy; accessed September 18, 2026.

Patrick P. Helm, Jan-Niklas Doerr, Joren Gijsbrechts, and Stefan Minner. Hard constraints, smooth gradients: Learning feasible inventory policies via differentiable projection, 2026. URL https://arxiv.org/abs/2608.02343v1. Version 1.

Luc Mercier and Pascal Van Hentenryck. Performance analysis of online anticipatory algorithms for large multistage stochastic integer programs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1979–1984, 2007.

Andrew Rosemberg, Alexandre Street, Davi M. Valladão, and Pascal Van Hentenryck. Efficiently training deep-learning parametric policies using Lagrangian duality, 2025. URL https://arxiv.org/abs/2405.14973v2. Version 2; first posted in 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Xianliang Yang, Zhihao Liu, Wei Jiang, Chuheng Zhang, Li Zhao, Lei Song, and Jiang Bian. A versatile multi-agent reinforcement learning benchmark for inventory management, 2023. URL https://arxiv.org/abs/2306.07542v1. Version 1.

Tinghan Ye, Shuaicheng Tong, Changkun Guan, Beste Basciftci, and Pascal Van Hentenryck. Learning optimization proxies for sequential contextual stochastic programs: An order fulfillment application, 2026. URL https://arxiv.org/abs/2606.25362v1. Version 1.

Enpeng Yuan, Wenbo Chen, and Pascal Van Hentenryck. Reinforcement learning from optimization proxy for ride-hailing vehicle relocation. *Journal of Artificial Intelligence Research*, 75:985–1002, 2022. doi: 10.1613/jair.1.13794.

Haoruo Zhao, Mathieu Tanneau, and Pascal Van Hentenryck. On the viability of stochastic economic dispatch for real-time energy market clearing, 2024. URL https://arxiv.org/abs/2308.06386v2. Version 2; first posted in 2023.

A DISPATCH FORMULATION

The formulation describes the two-stage stochastic dispatch program used as the optimization baseline at one fixed current time t. To simplify the LP, the outer index t is suppressed; τ always denotes a future look-ahead period. This is the optimization model used for baseline solves and stored comparison actions, not an optimization subproblem in the rollout loss.

Let $\mathcal{G}$, $\mathcal{N}$, $\mathcal{L}$, $\mathcal{S}$, $\mathcal{K}$, and $\mathcal{A}$ be the generator, bus, line, future-scenario, reserve-product, and reserve-area sets, and let $\mathcal{G}_n \subseteq \mathcal{G}$ be the generators connected to bus n. The current first-stage node is $i = 0$. A future recourse node is $i = (\tau, s)$ with $\tau \in \{1, \dots, H-1\}$ and $s \in \mathcal{S}$. Define

$$\mathcal{I}^+ = \{(\tau, s) : \tau = 1, \dots, H-1,\ s \in \mathcal{S}\}, \qquad \mathcal{I} = \{0\} \cup \mathcal{I}^+, \qquad w_i = \begin{cases} 1, & i = 0, \\ \rho_s, & i = (\tau, s), \end{cases}$$

where $\sum_s \rho_s = 1$. Define $p_{g,\mathrm{pred}(i)}$ as p_g^{prev} for $i = 0$, $p_{g,0}$ for $i = (1, s)$, and $p_{g,\tau-1,s}$ for $i = (\tau, s)$ with $\tau > 1$. A single current node rather than scenario copies enforces first-stage nonanticipativity by construction.

For node $i \in \mathcal{I}$, let $p_{g,i}$ be dispatch, $r_{m,g,i}$ reserve, $q_{n,i}$ net bus injection, and $c_{g,i}$ the epigraph variable for piecewise-linear generation cost. The two-sided line slack $\xi^{\mathrm{line}}_{\ell,i}$ is defined at every node. Load shed $\ell^{\mathrm{shed}}_{n,i}$, energy surplus $\ell^{\mathrm{sur}}_{n,i}$, and product-area reserve shortfall $\xi^{\mathrm{res}}_{m,a,i}$ are defined only for future nodes $i \in \mathcal{I}^+$. Their absence at $i = 0$ makes first-stage energy balance and reserve satisfaction hard. Let $a(g)$ be generator g's area, $e_{m,g} \in \{0, 1\}$ its eligibility for product m, $m_{m,a} \in \{0, 1\}$ indicate whether product m is required in area a, and $d_m \in \{-1, +1\}$ be reserve direction. The five products in the generated data are `Spin_Up_R1`, `Spin_Up_R2`, `Spin_Up_R3`, `Reg_Up`, and `Reg_Down`.

The implemented first-stage solution map returns

$$x_0^\star(I) = \left((p^\star_{g,0})_{g\in\mathcal{G}}, (r^\star_{m,g,0})_{m\in\mathcal{K}, g\in\mathcal{G}}\right), \tag{12}$$

where I denotes the complete current SLAD instance. The current temporal network predicts dispatch logits only; repair returns this dispatch-and-reserve action type. The baseline solution map is not a supervision target for rollout training, which evaluates repaired actions through their sequential operating costs.

Let $\mathcal{J}_g$ index the active affine pieces $(\alpha_{g,j}, \beta_{g,j})$ of generator g's production cost. Let $c^{\mathrm{res}}_{m,g}$ be its reserve offer, $D_{n,i}$ demand, $R_{m,i}$ the product requirement, $\underline{p}_{g,i}$ and $\overline{p}_{g,i}$ solver-facing dispatch bounds, $\Delta_g^\uparrow$ and $\Delta_g^\downarrow$ the five-minute ramp limits, $\Phi_{\ell n}$ the PTDF matrix, and $\overline{F}_\ell$ the line limit. The duration Δt is measured in hours. Penalties M^{E}, M^{res}_m, and M^{line} price energy imbalance, reserve shortfall, and line overload, respectively. The finite-scenario LP used by the hard-first-stage direct optimization baseline is

$$\begin{aligned}
\min \quad & \Delta t \Bigg\{ \sum_{i\in\mathcal{I}} w_i \left[\sum_{g\in\mathcal{G}} c_{g,i} + \sum_{m\in\mathcal{K}} \sum_{g\in\mathcal{G}} c^{\mathrm{res}}_{m,g} r_{m,g,i} + M^{\mathrm{line}} \sum_{\ell\in\mathcal{L}} \xi^{\mathrm{line}}_{\ell,i} \right] \\
& \qquad + \sum_{i\in\mathcal{I}^+} w_i \left[M^{\mathrm{E}} \sum_{n\in\mathcal{N}} \left(\ell^{\mathrm{shed}}_{n,i} + \ell^{\mathrm{sur}}_{n,i}\right) + \sum_{m\in\mathcal{K}} \sum_{a\in\mathcal{A}} M^{\mathrm{res}}_m \xi^{\mathrm{res}}_{m,a,i} \right] \Bigg\} && \text{(13a)} \\
\text{s.t.} \quad & c_{g,i} \geq \alpha_{g,j} p_{g,i} + \beta_{g,j}, \qquad \forall g, i,\ j \in \mathcal{J}_g, && \text{(13b)} \\
& q_{n,0} = \sum_{g\in\mathcal{G}_n} p_{g,0} - D_{n,0}, \qquad \forall n, && \text{(13c)} \\
& q_{n,i} = \sum_{g\in\mathcal{G}_n} p_{g,i} + \ell^{\mathrm{shed}}_{n,i} - \ell^{\mathrm{sur}}_{n,i} - D_{n,i}, \qquad \forall n, i \in \mathcal{I}^+, && \text{(13d)} \\
& \sum_{n\in\mathcal{N}} q_{n,i} = 0, \qquad \forall i, && \text{(13e)} \\
& \underline{p}_{g,i} \leq p_{g,i} \leq \overline{p}_{g,i}, \qquad \forall g, i, && \text{(13f)} \\
& p_{g,i} - p_{g,\mathrm{pred}(i)} \leq \Delta_g^\uparrow, \qquad \forall g, i, && \text{(13g)}
\end{aligned}$$

$$p_{g,\mathrm{pred}(i)} - p_{g,i} \le \Delta_g^{\downarrow}, \qquad \forall g, i, \tag{13h}$$

$$\sum_{g\in\mathcal{G}:\ a(g)=a,\ e_{m,g}=1} r_{m,g,0} = R_{m,0}, \qquad \forall m, a : m_{m,a} = 1, \tag{13i}$$

$$\sum_{g\in\mathcal{G}:\ a(g)=a,\ e_{m,g}=1} r_{m,g,i} + \xi^{\mathrm{res}}_{m,a,i} = R_{m,i}, \qquad \forall i \in \mathcal{I}^+, m, a : m_{m,a} = 1, \tag{13j}$$

$$0 \le r_{m,g,i} \le e_{m,g}\overline{p}_{g,i}, \qquad \forall i, m, g, \tag{13k}$$

$$p_{g,i} + \sum_{m:d_m=+1} r_{m,g,i} \le \overline{p}_{g,i}, \qquad \forall g, i, \tag{13l}$$

$$p_{g,i} - \sum_{m:d_m=-1} r_{m,g,i} \ge \underline{p}_{g,i}, \qquad \forall g, i, \tag{13m}$$

$$-\overline{F}_\ell - \xi^{\mathrm{line}}_{\ell,i} \le \sum_{n\in\mathcal{N}} \Phi_{\ell n} q_{n,i} \le \overline{F}_\ell + \xi^{\mathrm{line}}_{\ell,i}, \qquad \forall \ell, i, \tag{13n}$$

$$0 \le \ell^{\mathrm{shed}}_{n,i} \le D_{n,i}, \qquad \forall n, i \in \mathcal{I}^+, \tag{13o}$$

$$q_{n,i} \in \mathbb{R}, \qquad \forall n, i, \tag{13p}$$

$$c_{g,i}, \xi^{\mathrm{line}}_{\ell,i} \ge 0, \qquad \forall g, \ell, i, \tag{13q}$$

$$\ell^{\mathrm{sur}}_{n,i}, \xi^{\mathrm{res}}_{m,a,i} \ge 0, \qquad \forall \text{ valid } n, m, a,\ i \in \mathcal{I}^+. \tag{13r}$$

Here $p_{g,\mathrm{pred}(0)} = p_g^{\mathrm{prev}}$.

The hard-first-stage solver path used for the recorded comparison omits first-stage load shed, energy surplus, and reserve shortfall. Consequently, 13c and 13i enforce the implemented action's energy balance and reserve requirements without non-thermal slack. Future scenario nodes retain the penalized recourse variables in 13d and 13j. Transmission remains soft at every node through 13n, while physical dispatch bounds, interperiod ramping, reserve eligibility, and reserve headroom or footroom remain hard at every node. Reserve is not aggregated: every active product-area pair is enforced separately. The current data assign zero reserve- offer slopes because RTS-GMLC provides no reserve bids, so there is no direct reserve-procurement cost. Reserve still changes dispatch opportunity cost through headroom/footroom, and reserve shortfall is penalized where future recourse slack is allowed. The implementation retains the general reserve-cost term for data sets with reserve bids. The penalties are $M^{\mathrm{E}} = \$100{,}000/\mathrm{MWh}$, $M^{\mathrm{res}}_m = \$50{,}000/\mathrm{MW}$ for spinning products or $\$55{,}000/\mathrm{MW}$ for regulation products, and $M^{\mathrm{line}} = \$1{,}500/\mathrm{MW}$.

B Dispatch Repair

Repair runs in two configurations, and Algorithm 1 covers both. Training uses the soft-line configuration: it enforces balance, bounds, ramping, and reserve, leaves line overload to the penalty in the training loss, and performs no thermal redispatch. Deployment uses the complete configuration: it adds thermal redispatch and an independent audit of every constraint, so implemented actions satisfy the line limits as well.

All quantities below refer to one current instance. The reserve-band construction uses the reserve structure of RTS-GMLC, in which eligible-generator groups are disjoint and all active products in an area share the same eligible set; the implementation verifies this structure. Generator bounds include the availability and commitment information in the current instance.

Ramp-aware proposal. For generator g, let $\underline{p}_g$ and $\overline{p}_g$ be its physical limits, p_g^{prev} its previous implemented dispatch, and $\Delta_g^{\downarrow}$ and $\Delta_g^{\uparrow}$ its five-minute ramp limits. The effective bounds and bounded proposal are

$$L_g = \max\{\underline{p}_g, p_g^{\mathrm{prev}} - \Delta_g^{\downarrow}\}, \qquad U_g = \min\{\overline{p}_g, p_g^{\mathrm{prev}} + \Delta_g^{\uparrow}\}, \tag{14}$$

$$u_g = L_g + (U_g - L_g)\,\mathrm{sigmoid}(\hat{u}_g), \tag{15}$$

where $\hat{u}_g$ is the raw dispatch logit and u_g is the decoded dispatch proposal in MW. In the main formulation, f_θ includes this output map, so $u_t = f_\theta(I_t)$ denotes the proposal after decoding.

For vectors satisfying $l \le v \le u$ and a feasible target $a \in [\mathbf{1}^\top l, \mathbf{1}^\top u]$, define

$$\mathrm{Interp}(v; l, u, a) = \begin{cases} v + \dfrac{a - \mathbf{1}^\top v}{\mathbf{1}^\top u - \mathbf{1}^\top v}(u - v), & \mathbf{1}^\top v < a, \\ v + \dfrac{\mathbf{1}^\top v - a}{\mathbf{1}^\top v - \mathbf{1}^\top l}(l - v), & \mathbf{1}^\top v > a, \\ v, & \mathbf{1}^\top v = a. \end{cases} \tag{16}$$

This map moves proportionally toward the available upper or lower capacity and returns a vector whose sum is a, up to numerical tolerance.

Balance and reserve. Write the disjoint eligible sets as $\mathcal{G}_1, \ldots, \mathcal{G}_A$, where A is the number of active reserve areas. The remaining generators form $\mathcal{G}_0$. Let $m \in \mathcal{K}$ index reserve products, R_m be product m's requirement, and $d_m \in \{-1, +1\}$ its direction. For area $a > 0$, let $R_a^\uparrow$ and $R_a^\downarrow$ be the sums of its upward and downward product requirements. Its feasible energy band is

$$A_a = \max\left\{ \sum_{g \in \mathcal{G}_a} L_g, \sum_{g \in \mathcal{G}_a} \underline{p}_g + R_a^\downarrow \right\}, \tag{17}$$

$$B_a = \min\left\{ \sum_{g \in \mathcal{G}_a} U_g, \sum_{g \in \mathcal{G}_a} \overline{p}_g - R_a^\uparrow \right\}. \tag{18}$$

For $\mathcal{G}_0$, $A_0 = \sum_{g \in \mathcal{G}_0} L_g$ and $B_0 = \sum_{g \in \mathcal{G}_0} U_g$. With $Q_a = \sum_{g \in \mathcal{G}_a} u_g$ and $\bar{Q}_a = \mathrm{clip}(Q_a, A_a, B_a)$, demand D is balanced first across groups and then within each group:

$$(P_0, \ldots, P_A) = \mathrm{Interp}((\bar{Q}_a)_{a=0}^A; (A_a)_{a=0}^A, (B_a)_{a=0}^A, D), \tag{19}$$

$$(p_g)_{g \in \mathcal{G}_a} = \mathrm{Interp}((u_g)_{g \in \mathcal{G}_a}; (L_g)_{g \in \mathcal{G}_a}, (U_g)_{g \in \mathcal{G}_a}, P_a). \tag{20}$$

For an eligible upward or downward product m in area a, reserve is then reconstructed as

$$r_{m,g} = R_m \frac{\overline{p}_g - p_g}{\sum_{j \in \mathcal{G}_a} (\overline{p}_j - p_j)}, \qquad d_m = +1, \tag{21}$$

$$r_{m,g} = R_m \frac{p_g - \underline{p}_g}{\sum_{j \in \mathcal{G}_a} (p_j - \underline{p}_j)}, \qquad d_m = -1. \tag{22}$$

Ineligible coordinates are zero. At a zero requirement and zero available capacity, reserve is set to zero rather than forming an undefined ratio. Equations 17– 22 therefore enforce demand balance, each product-area requirement, eligibility, and shared directional capacity. Reserve is a deterministic function of repaired dispatch, physical headroom/footroom, requirements, and eligibility; it is not predicted by the temporal network.

Globally informed thermal completion. Let Φ be the PTDF matrix, d_b demand at bus b, $b(g)$ generator g's bus, and $\overline{F}_\ell$ line ℓ's symmetric limit. Its flow and overload are

$$f_\ell(p) = \sum_b \Phi_{\ell b} \left(\sum_{g : b(g) = b} p_g - d_b \right), \qquad v_\ell(p) = \left[|f_\ell(p)| - \overline{F}_\ell \right]_+. \tag{23}$$

Define the global violation merit

$$\mathcal{M}(p) = \left(\max_\ell v_\ell(p), \sum_\ell v_\ell(p) \right), \tag{24}$$

ordered lexicographically: a candidate is accepted when it reduces the maximum overload by more than $\epsilon_{\mathrm{merit}}$, or does not increase that maximum and reduces total overload by more than $\epsilon_{\mathrm{merit}}$. This check prevents a transfer that relieves the selected line from silently worsening the network-wide violation state.

Algorithm 1 Structured repair with global thermal-merit awareness

Require: Raw dispatch logits $\hat{u}$; instance I; configuration $\kappa \in \{\mathrm{soft}, \mathrm{complete}\}$

Compute L, U and the decoded proposal u using Equations 14–15.
Build reserve-area bands $[A_a, B_a]$ using Equations 17–18.
Balance group totals to demand, then balance generators within each group using Equations 19–20.
Reconstruct every reserve product using Equations 21–22.
if $\kappa = \mathrm{complete}$ **then**
  while $\max_\ell v_\ell(p) > \epsilon_{\mathrm{line}}$ and fewer than 512 transfers **do**
    Select the most violated oriented line and its sensitivities h_g.
    Form the maximum-relief pair and its capacity-capped transfer.
    if that transfer improves $\mathcal{M}(p)$ **then**
      Accept it.
    else
      For each full or backtracked factor, select the pair with smallest candidate $\mathcal{M}$; accept the first selected pair that passes the improvement test.
    end if
  end while
  Reconstruct reserve for the thermally redispatched p.
end if
Audit balance, bounds, ramping, reserve, and, for complete repair, every line limit.
return $\tilde{x} = (p, r)$

The repair repeatedly selects the worst oriented overload in Equation 23 and transfers equal MW from a generator with higher signed PTDF sensitivity to one with lower sensitivity. A same-area transfer preserves its group total; a cross-area transfer is capped by both groups' remaining energy-band capacity. Every transfer is also capped by the source's downward and sink's upward ramp-aware room. It therefore preserves balance and the conditions needed to reconstruct reserve.

For a selected oriented line, h_g is generator g's signed PTDF sensitivity and $\bar{\delta}_{ij}$ is the largest feasible transfer from source i to sink j under the effective bounds and reserve-area bands. The inexpensive first candidate maximizes $(h_i - h_j)\bar{\delta}_{ij}$. If that candidate does not improve Equation 24, the method evaluates all feasible relieving pairs, first at full transfer and then with backtracking factors $\{1/2, 1/4, 1/8, 1/16\}$. At each factor it selects the pair with the lexicographically smallest candidate merit and applies the improvement test to that pair, and it accepts the first pair that passes the test. The positive tolerances ϵ_{line}, $\epsilon_{\mathrm{solve}}$, and $\epsilon_{\mathrm{merit}}$ govern line acceptance, numerical over-correction, and merit improvement, respectively.

Every accepted transfer strictly decreases the global merit, and the number of transfers is bounded by the transfer cap. Line and generator-pair selection are discrete; after those choices, accepted transfer amounts remain ordinary Torch arithmetic and are differentiable piecewise. The repair solves no optimization problem. Every action returned by complete repair is feasible: balance, bounds, ramping, and reserve hold by construction, and every constraint, including every line limit, is certified by the independent audit before the action is returned.

Tolerances. The feasibility and line tolerances are 10^{-6} MW, the numerical correction and merit tolerances are 10^{-8} MW, and the transfer cap is 512.

C Training and Data Details

Throughout this appendix, *epoch* refers to a training epoch, that is, one pass over the training episodes; decision epochs within an episode are indexed by t as in Section 3.

C.1 Daily data and initial conditions

The daily dispatch data are derived from RTS-GMLC real-time streams. Commitment and online capacity are supplied by the daily unit-commitment pipeline. A SCED warm-up supplies the saved

previous-dispatch state at 01:00; the modeled training episode then contains 276 five-minute decisions. These setup solves occur before training iterations. Each day starts from its own saved state, and the learner replays the day's exogenous information while producing its own subsequent dispatch sequence.

The configuration uses $H = 12$, $S_t = 10$, and chronological 70/15/15 fractions, with complete-day validation and a horizon embargo at split boundaries. The scenario and forecast inputs are fixed exogenous data during a training rollout; gradients flow through the action-dependent state and associated features, not through the historical scenario-generation procedure. Future realizations used for the episode loss are not exposed to the current action beyond the forecasts available at that time.

Optimization supplies the daily initial state and marks which recorded instances are eligible; the learning objective uses policy-induced costs alone.

C.2 Data partitioning and training configuration

The data of each case study are partitioned chronologically into training, validation, and test blocks in a 70/15/15 split.

Energy dispatch. Training instances are complete daily episodes of $T = 276$ decisions, and look-ahead windows that cross split boundaries are excluded; input normalization uses training data only. The per-step loss is the generation cost plus the line-overload penalty $P_\kappa = \mu_{\mathrm{th}} \Delta t \, M^{\mathrm{line}} \sum_\ell v_\ell(\tilde{p}_{d,t})$, with the overload v_ℓ of Equation 23, the epoch length Δt, and $\mu_{\mathrm{th}} = 2$, and the movement term $\|\tilde{p}_{d,t} - u_{d,t}\|_2^2$ compares dispatch only, since reserves are reconstructed by repair. Batches contain up to ten days, the loss is scaled by 10^{-3}, and gradients are clipped at norm 10 before each Adam update. For each temporal regime, the learning rate is searched over five values between 2.5×10^{-6} and 10^{-4} and crossed with $\lambda_{\mathrm{move}} \in \{10^{-3}, 3 \times 10^{-3}, 10^{-2}\}$, with a 100-epoch budget. A scheduler halves the learning rate when training loss plateaus and fixes each candidate's terminal checkpoint; among candidates that are feasible at every validation decision, the lowest mean daily validation cost is selected. This gives learning rate 10^{-4} for the truncated and 5×10^{-5} for the fully unrolled configurations, with $\lambda_{\mathrm{move}} = 10^{-3}$; the $W = 12$, $W = 24$, and fully unrolled models stop at epochs 90, 85, and 90. The test block covers 55 days and 15,169 decisions: 54 days of 276 decisions and a final day of 265 decisions that ends at 23:00 so that the look-ahead stays within the block.

Inventory replenishment. The split gives 823 training, 176 validation, and 177 test days. Training uses $W = 64$, giving 13 parameter updates per epoch, with Adam at learning rate 3×10^{-4}, dropout 0.1, and a 75-epoch budget. The reported checkpoint is epoch 73, selected by mean daily validation profit.

C.3 Full rollout gradient

This appendix collects every dependency of the rollout objective on θ, of which Section 4.3 states one path. The policy Jacobians below are the action partial derivatives used there: $A_t = \partial \tilde{x}_t / \partial z_t$ and $B_t = \partial \tilde{x}_t / \partial \theta$.

Let A_t and B_t denote the policy's Jacobians with respect to the current state and parameters, respectively. At I_t, with $\kappa = \kappa_{\mathrm{tr}}$, the chain rule applied to Equation 7 gives

$$\begin{aligned} A_t &= \frac{\partial \pi_{\theta,\kappa}}{\partial z_t} = \frac{\partial \mathcal{R}_\kappa}{\partial u_t} \frac{\partial f_\theta}{\partial z_t} + \frac{\partial \mathcal{R}_\kappa}{\partial z_t}, \\ B_t &= \frac{\partial \pi_{\theta,\kappa}}{\partial \theta} = \frac{\partial \mathcal{R}_\kappa}{\partial u_t} \frac{\partial f_\theta}{\partial \theta}. \end{aligned} \tag{25}$$

B_t isolates the parameter effect at a fixed state, while $\partial \mathcal{R}_\kappa / \partial z_t$ isolates repair's state dependence at a fixed proposal. Thus A_t captures two paths of dependency from state to action: through the neural proposal and through the constraints used by repair. Let $Z_t = \mathrm{d}z_t / \mathrm{d}\theta$ denote the state sensitivity. Define the transition Jacobians by

$$F_{z,t} = \frac{\partial F_t}{\partial z_t}, \qquad F_{x,t} = \frac{\partial F_t}{\partial x_t},$$

evaluated at $(z_t, \tilde{x}_t, h_{t+1}^{\text{exo}})$ with the other arguments held fixed. They measure the next state's response to changes in the current state and action, respectively. Since z_t and $\tilde{x}_t$ both depend on θ, the transition's total derivative sums contributions through these two inputs. Applying the chain rule to the policy in Equation 7 and the state transition in Equation 2 gives

$$\begin{aligned} \frac{\mathrm{d}\tilde{x}_t}{\mathrm{d}\theta} &= \frac{\partial \pi_{\theta,\kappa}}{\partial \theta} + \frac{\partial \pi_{\theta,\kappa}}{\partial z_t}\frac{\mathrm{d}z_t}{\mathrm{d}\theta} = B_t + A_t Z_t, \\ Z_{t+1} &= \frac{\partial F_t}{\partial z_t}\frac{\mathrm{d}z_t}{\mathrm{d}\theta} + \frac{\partial F_t}{\partial x_t}\frac{\mathrm{d}\tilde{x}_t}{\mathrm{d}\theta} \\ &= F_{z,t} Z_t + F_{x,t}(B_t + A_t Z_t), \qquad Z_0 = 0. \end{aligned} \tag{26}$$

The term $A_t Z_t$ carries the indirect parameter effect through the evolving state. Each episode starts from a supplied initial state z_0 that is independent of θ, giving $Z_0 = 0$.

For the per-step training loss in Equation 9, let $\ell_{x,t}$, $\ell_{u,t}$, and $\ell_{z,t}$ denote its partial derivatives with respect to action, proposal, and state, evaluated at $(\tilde{x}_t, u_t; I_t)$ with the other arguments held fixed. The loss partial derivatives are row vectors that multiply Jacobians from the left. The proposal depends on θ directly and through z_t, so

$$\frac{\mathrm{d}u_t}{\mathrm{d}\theta} = \frac{\partial f_\theta}{\partial \theta} + \frac{\partial f_\theta}{\partial z_t} Z_t.$$

Substituting this derivative and Equation 26 into the chain rule gives the loss gradient:

$$\begin{aligned} \frac{\mathrm{d}\ell_\kappa(\tilde{x}_t, u_t; I_t)}{\mathrm{d}\theta} &= \ell_{x,t}\frac{\mathrm{d}\tilde{x}_t}{\mathrm{d}\theta} + \ell_{u,t}\frac{\mathrm{d}u_t}{\mathrm{d}\theta} + \ell_{z,t} Z_t \\ &= \ell_{x,t}(B_t + A_t Z_t) \\ &\quad + \ell_{u,t}\left(\frac{\partial f_\theta}{\partial \theta} + \frac{\partial f_\theta}{\partial z_t} Z_t\right) + \ell_{z,t} Z_t. \end{aligned} \tag{27}$$