Machine Learning–Enhanced Column Generation for Large-Scale Capacity Planning Problems

Felipe Keim[a], Víctor Bucarey[b,c], Qi Zhang[d], Angela Flores-Quiroz[a,c,*]

[a] *Departamento de Ingeniería Eléctrica, Universidad de Chile, Santiago, Chile*
[b] *Institute of Engineering Sciences, Universidad de O'Higgins, Rancagua, Chile*
[c] *Instituto Sistemas Complejos de Ingeniería, Santiago, Chile*
[d] *Department of Chemical Engineering and Materials Science, University of Minnesota,Minneapolis, MN 55455, USA*

Abstract

Capacity Planning problems are a class of optimization problems used in diverse industries to improve resource allocation and make investment decisions. Solving real-world instances of these problems typically requires significant computational effort. To tackle this, we propose machine-learning-aided column generation methods for solving large-scale capacity planning problems. Our goal is to accelerate column generation by approximating the pricing subproblems while preserving the ability to certify solution quality. We investigate two strategies embedded within the column generation framework: (i) a surrogate-based pricing approach that replaces the operational component of the pricing problem with a pre-trained multi-layer neural network with ReLU activations, incorporated through a mixed-integer linear (MILP) encoding of the network; and (ii) end-to-end approaches that learn to directly propose new columns. To retain performance guarantees, we adopt a two-phase procedure: an initial surrogate phase rapidly generates valid columns and approximate bounds, followed by a standard column generation phase that solves the original pricing subproblems to recover valid bounds and compute the optimality gap. Computational results show that the intermediate surrogate phase is beneficial in practice: it yields substantial runtime improvements, and the columns produced during the approximate phase are of sufficient

*Corresponding author
Email address: angflores@uchile.cl (Angela Flores-Quiroz)

quality to significantly speed up convergence in the subsequent exact phase.

Keywords: Column Generation, Machine Learning, Capacity Planning

1. Introduction

Capacity planning problems require making strategic decisions over long-term horizons—such as investments in new facilities, capacity upgrades, or storage assets—while simultaneously accounting for short-term operational constraints needed to satisfy demand that varies over time. A fundamental challenge in these problems is the presence of multiple temporal scales: long-term investment decisions, taken at coarse planning stages, must be consistently coordinated with fine-grained operational decisions. This interaction often leads to large-scale optimization models whose size and complexity grow rapidly with the temporal resolution of operational decisions.

A representative example arises in power system capacity expansion planning, where investment decisions must be integrated with short-term operational requirements such as unit commitment and dispatch over planning horizons that may span several decades [1, 2, 3, 4, 5, 6]. Despite this intrinsic complexity, capacity planning problems exhibit a structure that naturally lends itself to decomposition: while operational decisions and constraints are numerous, the constraints linking decisions across planning stages—typically associated with investment—are comparatively few. This property has motivated the use of decomposition techniques such as Progressive Hedging, Nested Benders decomposition, Stochastic Dual Dynamic Integer Programming, and Dantzig–Wolfe (DW) decomposition.

Within this context, column generation (CG) has emerged as a powerful approach for solving large-scale capacity planning problems [7, 8, 9]. However, in many applications, the pricing subproblems are computationally intensive, as they involve solving detailed operational models. Since CG does not require pricing subproblems to be solved to optimality at every iteration [10], reducing their computational burden can yield substantial performance gains. This observation motivates the use of machine learning models as a means to generate high-quality candidate columns at a lower

computational cost.

In this work, we investigate how neural networks can be leveraged to accelerate column generation for capacity expansion problems with challenging operational subproblems. We propose two ML-aided CG approaches. The first, replaces the pricing subproblem with a surrogate optimization model that approximates the operational component while preserving the structure of the subproblems and retaining perfomrance guarantees. The second, inspired by [11], adopts an end-to-end strategy in which a neural network directly predicts candidate columns and their reduced costs. We evaluate both approaches in a real-world case study on power system capacity expansion, and provide a comparative analysis of their computational performance, strengths, and limitations.

The contributions of this paper are:

- Proposal of a surrogate model of the pricing subproblem using a neural network to represent the operational part of the subproblem, exploiting the problem structure. This surrogate enables to generate valid columns with lower computational effort.
- Integration of the approximate solutions to the overall CG algorithm to maintain the validity of columns and bound, and achieve optimality.
- Inspired by the work of [11], we adapt, study, and compare an end-to-end Neural network approach that directly learns the solution of the pricing subproblem.
- In depth analysis of advantages and disadvantages of both methods.

2. Literature Review

Solving multi-stage capacity planning problems remains computationally demanding and typically requires tailored solution methods. A substantial body of literature has addressed two-stage and multi-stage capacity planning problems using decomposition-based approaches, including Progressive Hedging [12, 13], Nested Benders Decomposi-

tion [14, 15], Stochastic Dual Dynamic Integer Programming [16, 17], and Dantzig–Wolfe decomposition [9, 1, 7, 8]. These methods exploit the weak coupling between long-term investment decisions and short-term operational decisions to improve scalability.

Within this class of methods, column generation (CG) has shown strong computational performance for large-scale power system capacity planning problems, particularly when pricing subproblems involve integer decisions [1]. Nevertheless, in many capacity planning applications, pricing subproblems remain computationally expensive and often dominate the overall solution time. Importantly, CG does not require pricing subproblems to be solved to optimality at every iteration; any column with sufficiently negative reduced cost can improve the master problem. This observation opens the door to approximate solution strategies, motivating the use of machine learning techniques to reduce the computational burden of pricing while still generating high-quality columns.

An emerging line of research in operations research explores the integration of machine learning within decomposition algorithms, including column generation. One stream of work focuses on replacing the pricing subproblem with a learned model that directly predicts candidate columns. For instance, [18] employs reinforcement learning to learn a policy that outputs new columns for the cutting stock and vehicle routing problems, bypassing the explicit solution of the pricing problem. Similarly, [11] trains a transformer-based pointer network to generate job sequences with negative reduced cost for parallel machine scheduling, effectively approximating the pricing subproblem.

A second line of research aims at accelerating CG by predicting dual variables or improving algorithm initialization. In [19], [20], and [21], machine learning models are used to estimate dual prices to improve convergence and reduce oscillations in graph coloring, workforce scheduling, and cutting stock problems, respectively. Related approaches focus on warm-starting CG, as in [22], where ML models predict initial dual values for unit commitment problems. A third stream of work leverages ML to reduce the size or complexity of the pricing problem itself. Examples include [23] and [24], which use deep learning and graph neural networks to reduce pricing problem

size in personalized crew rostering and electric bus scheduling, respectively. In vehicle routing and bus driver scheduling, [25] employs reinforcement learning to select among pricing heuristics that generate reduced networks solved via optimization. Collectively, these studies demonstrate that ML-enhanced CG can achieve substantial reductions in computational time while maintaining solution quality.

Despite the growing body of work on integrating machine learning into column generation, important gaps remain when addressing large scale multi stage capacity planning problems with complex operational subproblems. Most existing approaches focus on combinatorial scheduling or routing settings, or rely on end to end predictions of pricing solutions without explicitly preserving the operational structure of the problem. In capacity expansion settings such as power system planning, pricing subproblems typically involve detailed operational models with network constraints, temporal coupling and integer decisions, making them particularly expensive to solve and difficult to approximate reliably. Moreover, only a limited number of studies address how to integrate machine learning into column generation while preserving valid bounds and convergence guarantees.

This work addresses these limitations by proposing two machine learning aided column generation approaches specifically tailored to multi stage capacity planning problems with complex operational subproblems. The first approach introduces a surrogate based pricing formulation in which the operational component of the pricing problem is approximated using a neural network, while the investment structure is kept explicit. This enables the generation of valid columns and the computation of certified optimality gaps through a two phase algorithm. The second approach adapts an end to end learning framework to directly predict pricing solutions for capacity planning problems, serving as a benchmark to evaluate the trade off between computational speed and solution guarantees. A detailed computational study on a realistic power system expansion problem is used to assess the performance, advantages and limitations of both approaches.

3. Column Generation for Solving Capacity Planning Problems

3.1. Multi-Stage Capacity Expansion Planning Problem

We consider a deterministic multi-stage capacity expansion problem defined over a discrete planning horizon. The problem consists of determining when and how much capacity to invest in, as well as how to operate the system over time to satisfy operational constraints and minimize total cost. We follow the general multi-stage capacity planning formulation proposed in [7] and focus on its deterministic counterpart.

The planning horizon is partitioned into a finite set of stages $\mathcal{S}$. For each stage $s \in \mathcal{S}$, the model includes investment decisions and operational decisions. Investment decisions, represented by integer variables $\boldsymbol{x}_s$, determine the set (or number) of capacity units added at stage s. The cumulative units available at stage s, accounting for all investments up to and including stage s, are represented by $\boldsymbol{z}_s$. Given $\boldsymbol{z}_s$, operational decisions $\boldsymbol{y}_s$ specify the system operation so as to satisfy system requirements and operational constraints.

Problem (1) defines the deterministic multi-stage capacity expansion problem. The objective function (1a) minimizes the total discounted cost over the planning horizon, including both investment and operational costs. In (1a), the vector $\boldsymbol{c}_s$ denotes the discounted investment cost at stage s, whereas $\boldsymbol{q}_s$ denotes the discounted operational cost at stage s.

Constraint (1b) couples decisions across stages by linking the cumulative number of available units $\boldsymbol{z}_s$ to the initial units $\bar{\boldsymbol{v}}$ and the investments made up to stage s. Constraint (1c) restricts operational decisions according to the number of units available at stage s, where matrices $\boldsymbol{A}_s$ and $\boldsymbol{V}_s$ relate operational decisions to unit availability. Constraint (1d) bounds the cumulative number of installed units at each stage, reflecting technological or policy constraints on capacity expansion. Constraint (1e) represents the set of operational constraints involving only the operational variables $\boldsymbol{y}_s$. Finally, constraint (1f) enforces the integrality of the investment decisions.

$$\min \sum_{s\in\mathcal{S}} \Big(\boldsymbol{c}_s^\top \boldsymbol{x}_s + \boldsymbol{q}_s^\top \boldsymbol{y}_s \Big) \tag{1a}$$

$$\text{s.t.: } \boldsymbol{z}_s \leq \bar{\boldsymbol{v}} + \sum\nolimits_{i=0}^{s} \boldsymbol{x}_i, \quad \forall\, s \in \mathcal{S}, \tag{1b}$$

$$\boldsymbol{A}_s \boldsymbol{y}_s \leq \boldsymbol{V}_s \boldsymbol{z}_s, \qquad \forall\, s \in \mathcal{S}, \tag{1c}$$

$$\boldsymbol{z}_s \leq \bar{\boldsymbol{z}}_s, \qquad \forall\, s \in \mathcal{S}, \tag{1d}$$

$$\boldsymbol{y}_s \in \mathcal{Y}_s, \qquad \forall\, s \in \mathcal{S}, \tag{1e}$$

$$\boldsymbol{x}_s \in \mathbb{Z}^+, \qquad \forall\, s \in \mathcal{S}. \tag{1f}$$

3.2. Dantzig-Wolfe reformulation of the Capacity Expansion Planning Problem

Problem (1) is decomposed using the Dantzig–Wolfe decomposition, following the discretization approach proposed in [7]. For each stage $s \in \mathcal{S}$, the set of feasible vectors of cumulative installed units $\boldsymbol{z}_s$ is defined as

$$\Psi_s = \left\{ \boldsymbol{z}_s \in \mathbb{Z}^+ \;\middle|\; \exists\, \boldsymbol{y}_s \in \mathcal{Y}_s \text{ such that } \boldsymbol{A}_s \boldsymbol{y}_s \leq \boldsymbol{V}_s \boldsymbol{z}_s,\ \boldsymbol{z}_s \leq \bar{\boldsymbol{z}}_s \right\}.$$

Since investment decisions are bounded and integer, Ψ_s is finite and can be enumerated as $\Psi_s = \{\widehat{\boldsymbol{z}}_s^k\}_{k\in\mathcal{K}_s}$, where $\mathcal{K}_s$ is cardinality of Ψ_s. Any feasible vector $\boldsymbol{z}_s \in \Psi_s$ is therefore represented as

$$\boldsymbol{z}_s = \sum_{k\in\mathcal{K}_s} \lambda_s^k \widehat{\boldsymbol{z}}_s^k, \tag{2a}$$

$$\sum_{k\in\mathcal{K}_s} \lambda_s^k = 1, \quad \lambda_s^k \in \{0,1\}. \tag{2b}$$

Moreover, for each cumulative investment vector $\widehat{\boldsymbol{z}}_s^k$, there exists at least one optimal operational plan $\widehat{\boldsymbol{y}}_s^k$. Substituting $\boldsymbol{z}_s$ and $\boldsymbol{y}_s$ in (1) using (2) yields the Dantzig–Wolfe

master problem (MP) in (3).

$$\min \sum_{s\in\mathcal{S}}\Big(\mathbf{c}_s^\top \boldsymbol{x}_s + \sum_{k\in\mathcal{K}_s} \boldsymbol{q}_s^\top \widehat{\boldsymbol{y}}_s^k \lambda_s^k\Big) \tag{3a}$$

$$\text{s.t. } : \sum_{k\in\mathcal{K}_s} \widehat{\boldsymbol{z}}_s^k \lambda_s^k \le \sum_{i=0}^{s} \boldsymbol{x}_i, \quad \forall\, s\in\mathcal{S}, \quad [\boldsymbol{\pi}_s] \tag{3b}$$

$$\sum_{k\in\mathcal{K}_s} \lambda_s^k = 1, \quad \forall\, s\in\mathcal{S}, \quad [\mu_s] \tag{3c}$$

$$\boldsymbol{x}_s \le \boldsymbol{b}_s, \quad \forall\, s\in\mathcal{S}, \tag{3d}$$

$$\boldsymbol{x}_s \in \mathbb{Z}^+, \quad \forall\, s\in\mathcal{S}, \tag{3e}$$

$$\lambda_s^k \in \{0,1\}, \quad \forall\, s\in\mathcal{S}\ \forall\, k\in\mathcal{K}_s. \tag{3f}$$

Problem (3) is solved using a column-generation (CG) algorithm [26], as outlined in Algorithm 1. The procedure starts by initializing a Restricted Master Problem (RMP) with the same structure as the master problem, but restricted to a limited subset of columns $(\widehat{\boldsymbol{z}}s^k, \widehat{\boldsymbol{y}}s^k)$. After solving the linear relaxation of the RMP, the dual variables $(\boldsymbol{\pi}s, \mu s)$ associated with constraints (3b) and (3c) are obtained. These dual variables are then used to identify improving columns by solving the pricing subproblem (4) for each stage $s \in \mathcal{S}$.

$$\text{sp}(s): \quad Z_s^{SP} = \quad \min \boldsymbol{q}_s^\top \boldsymbol{y}_s - \boldsymbol{z}_s \boldsymbol{\pi}_s - \mu_s \tag{4a}$$

$$\text{s.t.:} \qquad \boldsymbol{A}_s \boldsymbol{y}_s \le \boldsymbol{V}_s \boldsymbol{z}_s, , \tag{4b}$$

$$\boldsymbol{y}_s \in \mathcal{Y}_s, \tag{4c}$$

$$\boldsymbol{z}_s \le \bar{\boldsymbol{z}}_s, \boldsymbol{z}_s \in \mathbb{Z}^+. \tag{4d}$$

The subproblem is a one-stage investment and operation problem, where installed units $\boldsymbol{z}_s$ for stage s are determined. The objective of subproblem (4) is to find the column with most negative reduced cost, as described in (4a). Constraints (4b) and (4c) correspond to the operational constraints of stage s of the capacity planning problem,

and constraint (4d) impose the maximum number of units available for investment.

At each iteration of the algorithm, upper (UB) and lower (LB) bounds on the optimal value of the linear relaxation of problem (3) are computed. The upper bound is given by the optimal objective value of the current Restricted Master Problem (RMP), namely Z_{LP}^{MP}. The lower bound is obtained as $Z_{LP}^{MP} + \sum_{s \in \mathcal{S}} Z_s^{SP}$. Based on these bounds, the relative optimality gap for the linear relaxation is computed as $\frac{UB-LB}{LB}$. If this gap falls below a predefined tolerance, the integer master problem is subsequently solved to compute a feasible solution, together with the corresponding MIP optimality gap, and the algorithm terminates.

Algorithm 1 Column generation algorithm

1: Set $UB = \infty$, $LB = -\infty$, Initialize Master Problem (3) with initial set of columns
2: Solve the linear relaxation of Master Problem (3) and objective value Z_{LP}^{MP} and prices $(\boldsymbol{\pi}, \boldsymbol{\mu})$
3: Update upper bound:

$$UB \leftarrow Z_{LP}^{MP}$$

4: **for** $m \in \mathcal{M}$ **do**
5: Solve subproblem (4) with dual prices $(\boldsymbol{\pi}_m,\ \boldsymbol{\mu}_m)$, and collect objective value Z_m^{SP} and column $(\widehat{\boldsymbol{z}}_m, \widehat{\boldsymbol{y}}_m)$
6: Update lower bound:

$$LB \leftarrow Z_{LP}^{MP} + \sum_{s \in \mathcal{S}} Z_s^{SP}$$

7: Compute LP gap:

$$\delta_{LP} \leftarrow (UB - LB)/LB)$$

8: **if** $\delta_{LP} \leq \epsilon_{LP}$ **then**
9: Solve Integer Master Problem (3)
10: Collect objective value Z_{IP}^{MP} and compute MIP gap:

$$\delta_{IP} \leftarrow (Z_{IP}^{MP} - LB)/(LB)$$

11: **else**
12: go to 2

Note that, in capacity planning problems, the pricing subproblem itself can be a difficult MILP and is often the most computationally intensive component of the algorithm [1]. Therefore, reducing the time required to solve these subproblems can yield

a significant reduction in overall runtime of the algorithm. In the following section, we aim to overcome this computational burden by the approximation of the subproblems.

4. ML-enhanced heuristics for solving the pricing subproblem

This section presents an ML-enhanced column generation framework designed to reduce the computational burden of pricing subproblems while maintaining performance guarantees. The key observation is that, in column generation, it suffices to produce columns with negative reduced cost; consequently, the pricing subproblems need not be solved to optimality at every iteration. This insight enables the use of approximate, fast-to-evaluate pricing routines, which can substantially accelerate the overall algorithm.

Building on this idea, we investigate two ways to approximate the (computationally demanding) pricing subproblem using pre-trained neural networks. First, we propose a surrogate-based approach for column generation, where we replace the operational component of each pricing subproblem with a neural-network representation that estimates the operational cost as a function of the investment decisions and the problem structure. We then embed this estimator within an optimization-based surrogate problem. This constitutes the main contribution of our work and is detailed in Section 4.1. Second, Section 4.2 presents an alternative end-to-end strategy, inspired by [11], in which a neural network is trained to directly predict approximate solutions to the pricing subproblems.

4.1. Proposed Surrogate-based Column Generation

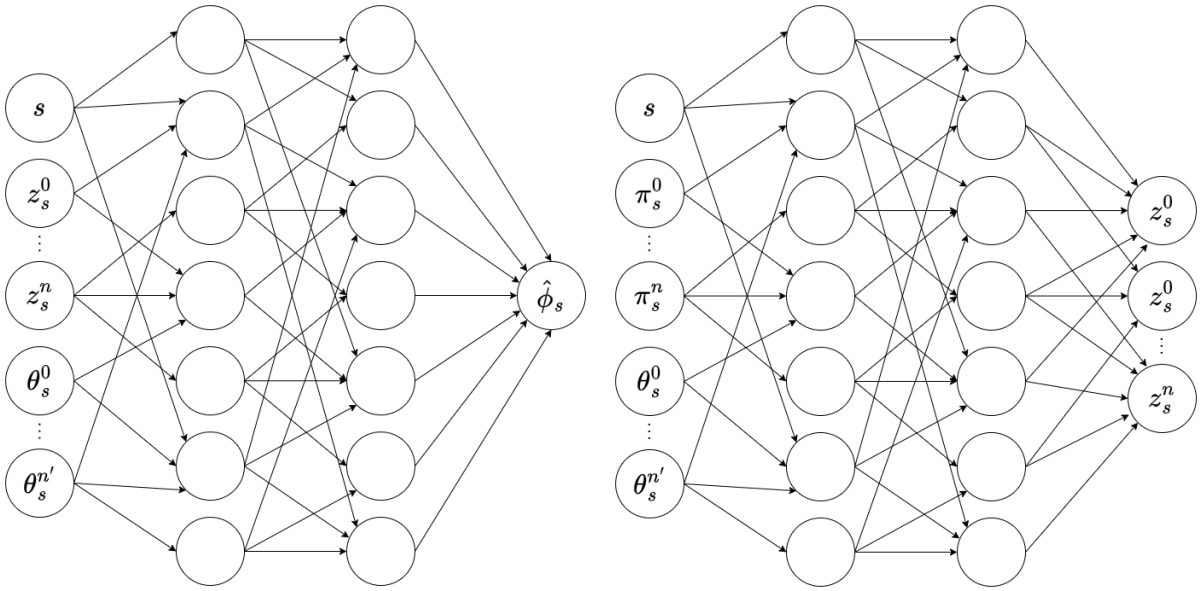
This section presents the proposed Surrogate-based column generation framework. Specifically, the operational component of the pricing subproblem is approximated using a neural network trained to predict operational costs as a function of installed capacity assets, which can then be embedded within the pricing problem. This surrogate-based approach significantly reduces computational time while preserving the quality of the generated columns. Importantly, the algorithm maintains performance guarantees through a two-phase procedure. In the first phase, surrogate-based pricing problems

are used to efficiently generate valid columns and compute approximate bounds. In the second phase, iterations with the original pricing subproblems are performed to recover valid lower bounds and compute a certified optimality gap. Section 4.1.1 describes the neural-network-based approximation of the pricing subproblem, while Section 4.1.2 explains how the surrogate model is integrated into the column generation algorithm following this two-phase approach.

4.1.1. A Surrogate Optimization Model for the Pricing Subproblem

We construct the surrogate pricing problem by approximating only the operational component of (4) (associated with $\boldsymbol{y}_s$), while keeping the investment variables $(\boldsymbol{x}_s, \boldsymbol{z}_s)$ and the constraint set unchanged. We learn a function that maps investment decisions to the corresponding optimal operational cost and embed it into the optimization model. Following [27], this function is represented by a pre-trained multi-layer ReLU network, trained to predict the optimal operational cost for a given investment vector. Using the mixed-integer linear encoding of ReLU networks from [28], we integrate the network into the pricing problem, obtaining a surrogate model that retains the original structure but is significantly cheaper to solve.

For each stage s, we define the operational cost $\phi_s(\boldsymbol{z}_s, \boldsymbol{\theta}_s)$ as the optimal objective value of the single-stage operational problem in (5), given the cumulative installed-capacity vector $\boldsymbol{z}_s$ and an instance-specific vector of correlated parameters $\boldsymbol{\theta}_s$. The parameters in $\boldsymbol{\theta}_s$ capture the information required to specify a given operational instance—namely, coefficients appearing in the objective function and constraints, as well as additional contextual features that may be useful for training. Accordingly, for a realization $\boldsymbol{\theta}_s$, we denote by $\boldsymbol{q}_s(\boldsymbol{\theta}_s)$, $\boldsymbol{A}_s(\boldsymbol{\theta}s)$, and $\Upsilon s, d(\boldsymbol{\theta}_s)$ the instantiated coefficients of the model. We then train a neural network to approximate the mapping $\boldsymbol{z}_s \mapsto \phi_s(\boldsymbol{z}_s, \boldsymbol{\theta}_s)$, yielding a predictor $\hat{\phi}_s(\boldsymbol{z}_s, \boldsymbol{\theta}_s)$, which is used to replace the operational component of the pricing subproblem. The network architecture, for n investment variables and n' parameters, is illustrated in Figure 1(a).



(a) Surrogate model Neural Network (b) End-to-End Neural Network

Figure 1: Two neural network structures for both approaches.

$$\phi_s(\boldsymbol{z}_s) = \quad \min \boldsymbol{q}_s(\boldsymbol{\theta})^\top \boldsymbol{y}_s \tag{5a}$$

$$\text{s.t.:} \qquad \boldsymbol{A}_s(\boldsymbol{\theta})\boldsymbol{y}_s \leq \bar{\boldsymbol{v}} + \boldsymbol{V}_s \boldsymbol{z}_s \tag{5b}$$

$$\boldsymbol{y}_{s,d} \in \Upsilon_{s,d}(\boldsymbol{\theta}) \tag{5c}$$

Once trained, the network is incorporated into the pricing subproblem using the MILP encoding of ReLU networks from [28]. This allows us to replace the operational block by the network while keeping the investment block intact: $\boldsymbol{z}_s$ and constraints (6i) are preserved, whereas $\boldsymbol{y}_s$, constraints (4b)–(4c), and the operational cost are replaced by the network output.

For a neural network with $\mathcal{M}$ layers, the MILP encoding introduces auxiliary continuous variables $\hat{h}_j^m$ and $\check{h}_j^m$, along with binary variables a_j^m, to model the ReLU activations through a mixed-integer formulation. Here, the parameters w and b of the neural network are obtained after training, so in the MILP representation, they are fixed. Constraints (6b)–(6h) encode the network and map the input vector $(s, \boldsymbol{z}_s, \boldsymbol{\theta})$ to

an estimate of the operational cost, where $[s, \boldsymbol{z}_s, \boldsymbol{\theta}]_i$ denotes the i^{th} component of the concatenated input. Including the stage index s enables a single network to be used throughout the planning horizon, while $\boldsymbol{\theta}$ promotes generalization in different system configurations.

$$\widehat{\text{sp}(s)}: \quad \min \hat{\phi}_s - \boldsymbol{z}_s \boldsymbol{\pi}_s - \mu_s \tag{6a}$$

$$\text{s.t.}: \sum_{i=1}^{d_0} w_{i,k}^0 \left[s, \boldsymbol{z}_s, \boldsymbol{\theta}\right]_i + b_j^0 = \hat{h}_j^1 - \check{h}_j^1, \quad \forall\, j \in \mathcal{D}_1, \tag{6b}$$

$$\sum_{i=1}^{d_{m-1}} w_{i,k}^{m-1} \hat{h}_j^{m-1} + b_j^{m-1} = \hat{h}_j^m - \check{h}_j^m, \quad \forall\, m \in \mathcal{M}, \quad \forall j \in \mathcal{D}_m, \tag{6c}$$

$$\sum_{i=1}^{d_\ell} w_{i,k}^{\ell} \hat{h}_j^{\ell} \leq \hat{\phi}_y, \tag{6d}$$

$$\hat{h}_j^m \leq M^+(1 - a_j^m), \quad \forall\, m \in \mathcal{M}, \quad \forall j \in \mathcal{D}_m, \tag{6e}$$

$$\check{h}_j^m \leq M^- a_j^m, \quad \forall\, m \in \mathcal{M}, \quad \forall j \in \mathcal{D}_m, \tag{6f}$$

$$\hat{h}_j^m, \check{h}_j^m \geq 0, \quad \forall\, m \in \mathcal{M}, \quad \forall j \in \mathcal{D}_m, \tag{6g}$$

$$a_j^m \in \{0, 1\}, \quad \forall\, m \in \mathcal{M}, \quad \forall j \in \mathcal{D}_m, \tag{6h}$$

$$\boldsymbol{z}_s \leq \bar{\boldsymbol{z}}_s, \boldsymbol{z}_s \in \mathbb{Z}^+. \tag{6i}$$

To train the neural network, we generate one dataset per stage. For each dataset, we sample feasible investment decisions and corresponding parameter values, compute their operational costs, and repeat until reaching the desired dataset size. All datasets are merged and used to train a ReLU-based neural network. Training is performed using an L1 loss function, dropout and early stopping to prevent overfitting, and the Adam optimizer. Hyperparameter are selected sequentially, first tuning the learning rate, then the network architecture, and finally the dropout probability.

4.1.2. Integrating Approximated Subproblems into Column Generation

The proposed surrogate-based column generation algorithm preserves performance guarantees through a two-phase procedure. Figure 2(a) illustrates the overall framework. In the first phase, surrogate pricing problems are used to quickly generate valid columns and compute *approximate bounds*. In the second phase, the algorithm switches to the original pricing subproblems to recover *valid bounds* and evaluate the optimality gap.

The proposed Surrogate-based column generation algorithm preserves performance guarantees through a two-phase procedure. Figure 2(a) illustrates the overall framework. In the first phase, surrogate pricing problems are used to quickly generate valid columns and compute *approximate bounds*. In the second phase, we switch to the original pricing subproblems to recover *valid bounds* and evaluate the optimality gap. To enable this two-phase scheme, the algorithm maintains two master problems. In the *first phase*, the Approximated Restricted Master Problem (ARMP), is used in the first phase and is built from columns generated by the surrogate pricing problems, together with their estimated operational costs. The *second phase* is the standard Restricted Master Problem (RMP), which contains only columns with exact operational costs and is used to compute valid bounds and certify optimality.

In the first phase, the ARMP is solved at each iteration to obtain dual variables. These dual prices are then used in the surrogate pricing problems to generate new columns. For each decision stage s, the surrogate pricing problem is solved to obtain a candidate investment vector $\hat{\boldsymbol{z}}_s$ together with an estimated operational cost $\hat{\phi}_s(\hat{\boldsymbol{z}}_s)$. The resulting columns are added to the ARMP. To generate valid columns for the RMP, the true operational cost $\phi_s(\hat{\boldsymbol{z}}_s)$ is evaluated for each candidate investment vector $\hat{\boldsymbol{z}}_s$. The corresponding columns, now associated with exact operational costs, are then added to the RMP. Since only the operational variables are optimized, this step is usually significantly faster than solving the full pricing subproblem.

Note that, within this first phase, the ARMP corresponds to the restricted master problem associated with an approximated version of the Dantzig–Wolfe master

problem (3), in which operational costs are represented through the surrogate model. Consequently, a valid lower bound for the linear relaxation of this approximated master problem can be obtained using the reduced costs from the surrogate pricing problems. In parallel, an upper bound $\widehat{UB}$ is obtained by solving the ARMP. Surrogate-based column generation iterations are performed until the optimality gap of the approximated master problem, denoted by $\widehat{LP}_{gap}$, falls below a predefined threshold. Since an approximation of the master problem is solved in the first phase (3), the resulting bounds and optimality gap provide estimates of the corresponding quantities for the original problem, and may not be valid for the latter.

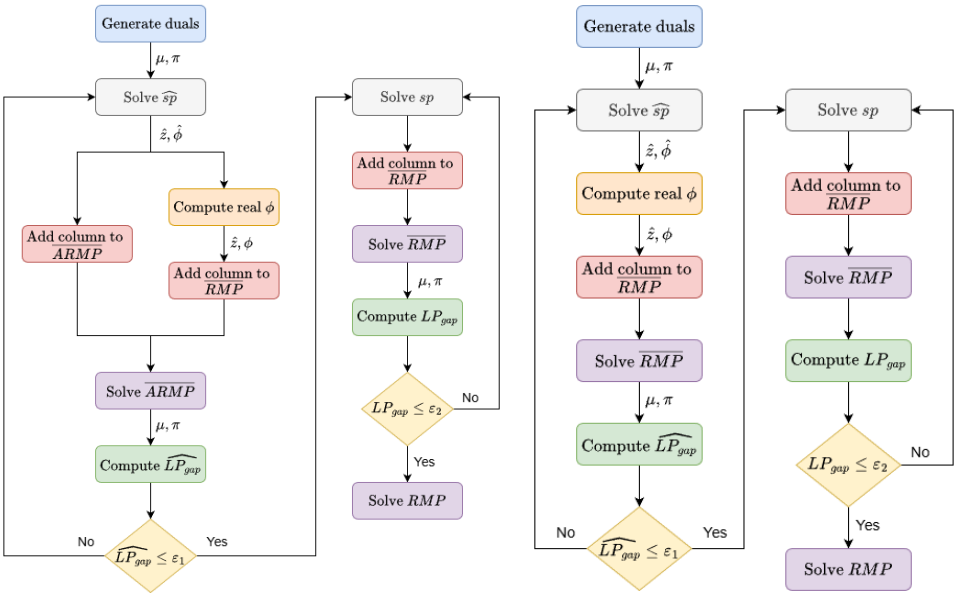
Thus, once this condition is met, the algorithm transitions to the second phase. This phase starts by solving the RMP populated with the valid columns generated during the first phase. New columns are then generated by solving the original pricing subproblem (4) and the lower and upper bounds are computed as in Algorithm 1. Thus, these exact CG iterations enable recovering valid upper and lower bounds and to compute a certified optimality gap. While these iterations are costly, in practice, only a small number of traditional CG iterations are required, as the first phase typically produces high-quality columns that are already close to optimal.

4.2. An End-to-End Approach

This section presents an End-to-End learning approach to accelerate column generation by directly approximating the pricing subproblems. Unlike the surrogate-based framework, which approximates only the operational cost while preserving the structure of the pricing problem, the End-to-End approach replaces each pricing subproblem with a neural network that directly predicts candidate investment decisions. As in the surrogate-based approach, performance guarantees are recovered by subsequently running iterations of the original column generation algorithm.

4.2.1. End-to-End Subproblem Approximation

In the End-to-End approach, each pricing subproblem is replaced by a neural network trained to directly predict an investment decision vector $\boldsymbol{z}_s$ for stage s. The



(a) Surrogate CG. (b) End-to-End CG.

Figure 2: Two frameworks for ML-Aided CG algorithms.

network takes as input the dual prices $(\boldsymbol{\pi}_s, \mu_s)$ obtained from the RMP, the stage index s, and a set of problem parameters $\boldsymbol{\theta}$, and outputs an investment vector. Figure 1(b) illustrates the structure of the End-to-End neural network.

Training data is generated by sampling dual prices and parameter values, and solving the corresponding pricing subproblems to optimality. Each training sample consists of dual prices, stage index, and parameters as inputs, and the optimal investment decision as the target output. The network is trained using a binary cross-entropy loss with logits, and hyperparameters are selected sequentially by tuning the learning rate, network architecture, and dropout probability.

This approach is inspired by the work proposed in [11], adapted to the capacity planning problem. Unlike the original work, which employs a transformer-based architecture, we adopt a feedforward ReLU-based neural network to predict subproblem solutions. We chose this option to keep the parsimony of the neural networks trained equal for both approaches. Additionally, instead of extracting dual prices from multiple column generation executions, we generate training data by sampling random dual price values from a uniform distribution.

4.2.2. Integrating the End-to-End NN into Column Generation

The End-to-End approach follows a two-phase procedure, analogous in spirit to the Surrogate-based column generation framework, in order to balance computational efficiency and solution quality. In contrast to the surrogate-based approach, only a single Restricted Master Problem (RMP) is maintained, which always contains valid columns evaluated with the true operational cost.

At each CG iteration of the first phase, the trained End-to-End neural network is used to predict a candidate investment vector $\hat{\boldsymbol{z}}_s$ for each stage s, given the current dual variables obtained from the RMP. To construct a valid column, the operational problem (5) is then solved for the predicted investment vector $\hat{\boldsymbol{z}}_s$, yielding the corresponding operational decision $\hat{\boldsymbol{y}}_s$ and its true operational cost. The column $(\hat{\boldsymbol{z}}_s, \hat{\boldsymbol{y}}_s)$ is subsequently added to the RMP, preserving primal feasibility.

Note that the upper bound obtained from the RMP remains valid throughout the End-to-End phase. However, a valid lower bound cannot be computed during this phase, as the neural network may predict suboptimal investment decisions. Instead, an estimated lower bound is obtained from the reduced costs associated with the predicted columns, yielding an estimated optimality gap.

End-to-End CG iterations are performed until this estimated gap falls below a predefined threshold. Since this stopping criterion is based on an estimated gap, it is possible that $\widehat{LP}_{gap}$ does not reach the prescribed tolerance. For this reason, a maximum number of iterations may be imposed as an alternative termination criterion. At that point, the algorithm transitions to the second phase, in which traditional column generation iterations are executed by solving the original pricing subproblems. This second phase recovers lower bounds and allows the computation of a certified optimality gap. Figure 2(b) illustrates the overall End-to-End CG procedure.

5. Computational Results and Discussion

This section applies the proposed methodologies to instances of a capacity expansion planning model for electric power systems. We consider a deterministic formulation of a power system expansion problem with operational constraints, based on the formulation presented in [1]. We report results on dataset generation, neural network training, and performance comparisons among the *traditional* column generation, the proposed it surrogate column generation, and the *end-to-end* benchmark approach.

All algorithms were implemented in Python v3.9.0. Optimization models were formulated using Pyomo v6.0.1 [29] and solved with Gurobi v9.5.0 [30]. Supervised learning models were trained using PyTorch v1.10.0 [31] and Scikit-learn v1.0.1 [32]. All computational experiments were executed on the NLHPC computing cluster.

5.1. Test System and Problem Setup

The case study considers a multi-stage expansion planning problem for an electrical power system. Investment is made in generation assets. The operation is represented by the unit commitment model. The test system is based on a realistic representation of the Chilean power system, consisting of 26 buses, 42 transmission lines, and 86 existing generators, with a planning horizon of 10 years. All data was obtained from the Chilean National Electric Coordinator (CEN) [33].

The objective is to minimize the total system cost, combining investment and operational costs, by jointly selecting investment and operational decisions over the planning horizon, subject to demand satisfaction and operational constraints. Demand evolves deterministically according to the growth rate R_{dem}. Load shedding is permitted at a high penalty cost, which guarantees the feasibility of the operational problem for any investment decision.

The problem considers binary investment variables accounting for the installation of new generation assets into the system, and mixed-integer operational variables, accounting for the hourly system operation. The operational variables include power outputs

of the generator, its commitment state, power flows through transmission lines, etc. To reduce the computational burden of computing the operation of each hour of the year, we select a set of representative days using the approach presented in [34]. The complete formulation of the generation and transmission planning problem is presented in Appendix A.

We consider different system configurations by varying the number of investment decisions and the number of representative operational days. A total of three configurations are analyzed, denoted as EPS_{X_Y}, where X represents the number of investment decisions and Y the number of representative days. Specifically, we study the configurations EPS_{15_4}, EPS_{15_12}, and EPS_{52_4}. Additionally, for both ML-based approaches, the demand growth rate of the system is used as the input parameter θ. The tested algorithm was subsequently evaluated under three demand growth scenarios for each network configuration: 3%, 5%, and 8%. Accordingly, each instance used to test the algorithm is denoted by $EPS_{X\_Y\_Z}$, where Z represents the demand growth rate.

5.2. Dataset Generation and Neural Network Training

For each network configuration EPS_{X_Y}, we generated dedicated datasets to train the neural networks associated with each ML-based approach by repeatedly solving optimization problems under different input configurations. We consider multilayer perceptron (MLP) architectures with ReLU activations. The number of hidden layers and neurons per layer, dropout rate, and learning rate are selected via hyperparameter tuning using a sequential grid search. Detailed results for each neural network are reported in Appendix B.

In the End-to-End approach, training samples are obtained by solving the pricing problem (4) for a wide range of sampled inputs. Specifically, for each decision stage s, dual variables $(\boldsymbol{\pi}_s, \mu_s)$ and system parameters θ are sampled, where θ corresponds to the demand growth rate. For each sampled input, the corresponding pricing problem is solved to optimality to obtain the investment decision $\boldsymbol{z}_s$. These optimal investment

vectors are used as labels to train a neural network that directly predicts candidate columns from dual information. The dual variables are sampled from a uniform distribution over the interval $[0, 500]$, while the demand growth rate is sampled from a uniform distribution between 1% and 7%.

In contrast, the surrogate approach does not require solving the full pricing problem during data generation. Instead, for fixed investment decisions $\boldsymbol{z}_s$, decision stages s, and demand growth rates, the operational problem (5) is solved to compute the corresponding optimal operational cost. These costs are used as training targets for a neural network that approximates the operational component of the pricing problem. During dataset generation, the demand growth rate is sampled from the same uniform distribution between 1% and 7%.

Using the resulting datasets, hyperparameter search and neural network training were performed for each network configuration EPS_{X_Y}. The average time required to generate a single training sample for each instance, as well as the total training time (including hyperparameter tuning plus training of the best network) for both the surrogate and End-to-End networks, are reported in Table 1.

Network configuration	Avg. Dataset Generation Time per Sample [s]		Total NN Training Time [s]	
	Surrogate	End-to-End	Surrogate	End-to-End
EPSP_A_15_4	44.21	109.16	621.42	601.28
EPSP_A_15_12	145.52	2184.18	506.66	483.85
EPSP_A_52_4	69.42	843.45	669.50	719.16

Table 1: Average time required to generate a single dataset sample and total neural network training time (in seconds) for each problem instance under both proposed approaches.

The computational cost per training sample differs substantially across approaches. Generating End-to-End training data requires solving the full pricing problem to optimality, whereas the surrogate approach only involves solving the operational subproblem. This leads to significantly lower data-generation times for the proposed surrogate-based approach.

For both approaches, we generated 5000 samples per decision stage, yielding a total of 50000 samples over the 10-stage planning horizon. Dataset generation was parallelized using up to 160 cores. While data generation is computationally intensive, this cost can be spread across multiple planning studies, since capacity expansion problems are typically solved repeatedly for scenario and sensitivity analyses with similar network structures and parameter ranges, allowing training datasets to be reused or incrementally extended. In contrast, neural network training times were relatively low compared to data generation times.

The results of the neural network performance are summarized in Tables 2 and 3 for the End-to-End and Surrogate approaches, respectively. For the End-to-End approach, the neural network predictions are evaluated in terms of accuracy, defined as the percentage of correct predictions for each investment candidate across the training, validation, and test sets. For the Surrogate approach, the performance of the neural network is measured using Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) on the training, validation, and test set.

For the Surrogate approach, prediction errors remain below 2% across all instances and across the training, validation, and test sets. Higher MAE values are observed for instances with 52 investment decisions, reflecting larger operational costs; however, relative errors remain small. Low prediction errors support the surrogate model in identifying negative reduced-cost columns, which can reduce the number of iterations required by the traditional column generation algorithm to reach the optimal solution. For the End-to-End approach, the neural network achieves an accuracy above 96% across all datasets.

Instance	MAE			MAPE [%]		
	Train	Valid	Test	Train	Valid	Test
EPSP_15_4	5.57	5.67	5.62	0.31	0.32	0.32
EPSP_15_12	5.92	6.04	7.14	0.33	0.33	0.35
EPSP_52_4	86.09	136.83	105.87	1.70	1.95	1.87

Table 2: Surrogate neural network prediction results: performance of the proposed NN across all problem instances for train, validation and test sets.

Instance	Accuracy [%]		
	Train	Valid	Test
EPSP_15_4	98.92	98.51	98.589
EPSP_15_12	98.78	98.44	98.455
EPSP_52_4	97.22	96.97	97.033

Table 3: End-to-End neural network prediction results: performance of the proposed NN across all problem instances for train, validation and test sets.

5.3. Performance Comparison

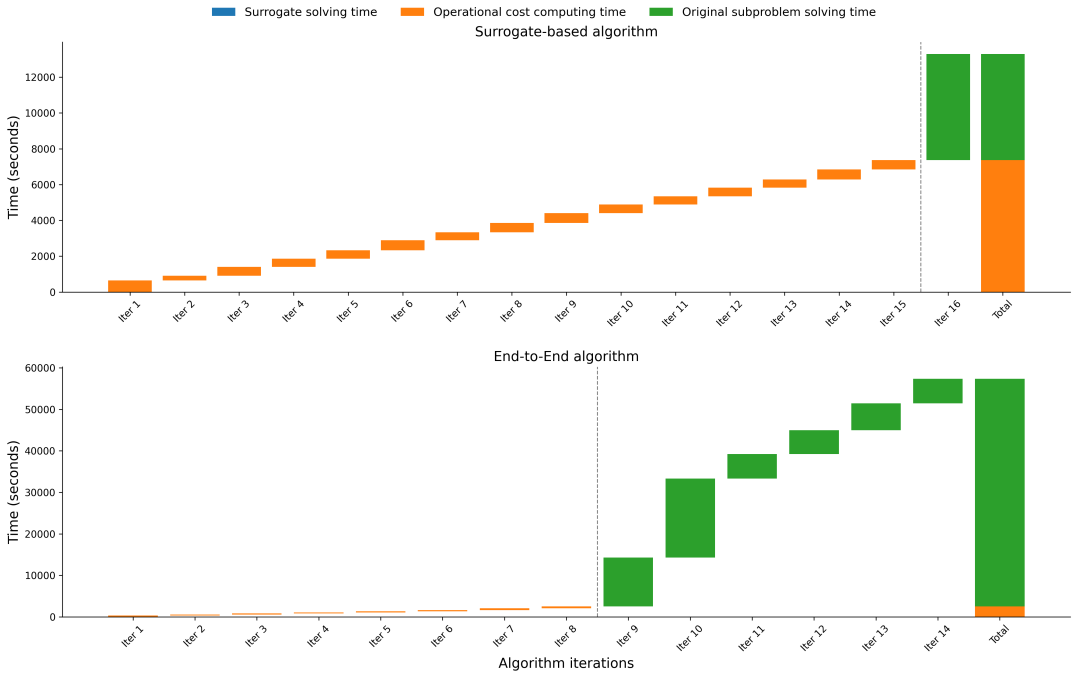
To evaluate performance across different system configurations, each problem instance $EPS_{X\_Y\_Z}$ was solved using three approaches: traditional column generation (CG), the proposed Surrogate-based CG, and the End-to-End CG. For all algorithms, an optimality gap of 0.5% was required. For the ML-based approaches, which follow a two-phase procedure, the first phase was terminated when an *estimated* optimality gap of $\varepsilon_1 = 0.25\%$ was reached. Afterwards, traditional CG iterations were executed until the final optimality gap of 0.5% was reached.

Table 4 reports the average computational time and the number of iterations per instance for the three algorithms. For the two ML-based approaches, the iteration count is reported as a tuple, with the number of ML-driven iterations and the number of subsequent traditional CG iterations. Overall, the proposed Surrogate-based CG achieves lower computational times than both traditional CG and the End-to-End CG across all tested instances.

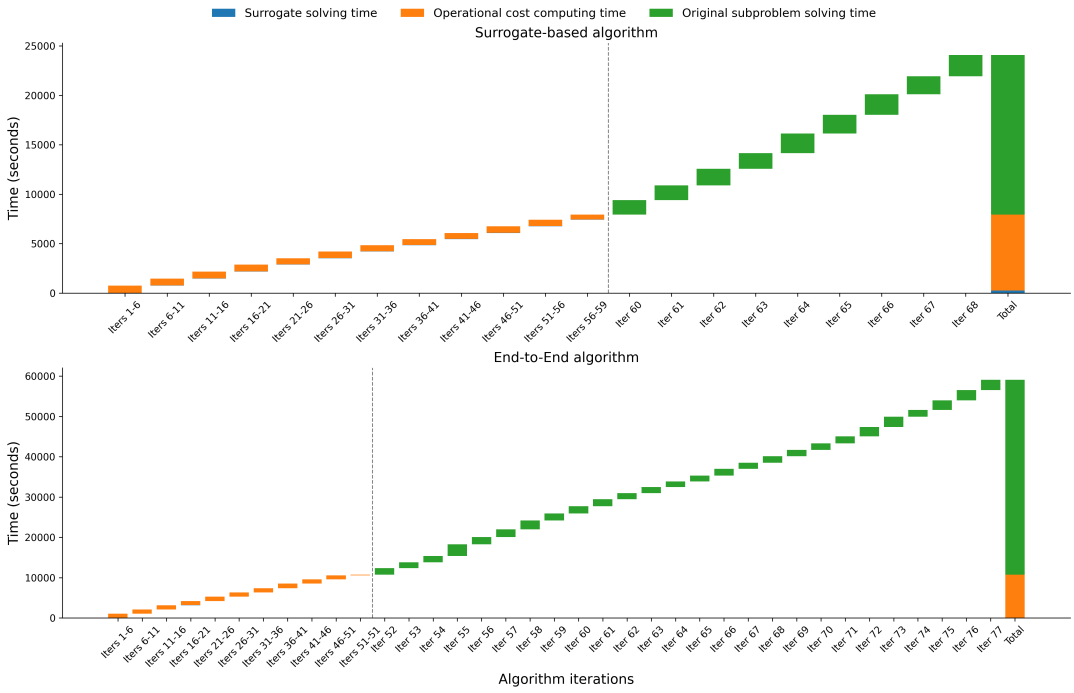
Instance	Rdem	Surrogate		End-to-End		Traditional	
		Time	Iters	Time	Iters	Time	Iters
EPSP_15_4	3%	00:14:44	(12, 1)	00:54:35	(12, 5)	01:08:42	10
EPSP_15_4	5%	00:15:55	(12, 1)	01:03:52	(12, 5)	01:12:41	10
EPSP_15_4	8%	00:46:40	(14, 3)	01:22:44	(4, 11)	01:15:05	11
EPSP_15_12	3%	02:12:04	(15, 1)	19:54:29	(6, 7)	1-05:06:15	10
EPSP_15_12	5%	03:42:34	(14, 1)	15:59:12	(8, 6)	1-03:35:35	12
EPSP_15_12	8%	04:52:32	(14, 3)	17:03:27	(4, 10)	20:18:49	12
EPSP_52_4	3%	06:43:22	(57, 9)	16:26:06	(50, 26)	1-08:09:48	58
EPSP_52_4	5%	10:18:46	(64, 9)	1-06:33:34	(8, 56)	2-04:00:18	52
EPSP_52_4	8%	16:24:06	(51, 16)	1-19:00:31	(4, 58)	1-10:16:18	51

Table 4: Three column generation Algorithms results in terms of number of iteration and time [dd-hh:mm:ss] for all instances and different demand growth rates.

The time required to predict the investment decision, either by solving the surrogate model or by evaluating the End-to-End neural network, is several orders of magnitude smaller than the time required to compute the true operational cost at each iteration. Similarly, solving the subproblems with fixed investment decisions is significantly faster than solving the complete subproblem. This behavior is illustrated in Figures 3(a) and 3(b), which show the cumulative runtime of the surrogate-based and End-to-End algorithms for two representative instances: $EPSP\_15\_12$ with a demand growth of 5% and $EPSP\_52\_4$ with a demand growth of 3%. The total computational time is decomposed into three main components: the time required to predict the investment plan using the machine learning model, the time required to compute the operational cost, and the time required to solve the original subproblems.



(a) Instance $EPSP\_15\_12$ with 5% demand growth.



(b) Instance $EPSP\_52\_4$ with 3% demand growth.

Figure 3: Cumulative runtime decomposition of the surrogate-based and End-to-End algorithms. Each bar represents the cumulative time spent in different algorithmic components, grouped by iterations.

Most of the computational effort of the algorithms is spent in solving traditional CG iterations. Consequently, the proposed Surrogate-based CG achieves lower overall computational time across all tested instances by requiring fewer traditional CG iterations to reach the optimality threshold. For instances with 15 investment candidates, the algorithm requires three or fewer traditional CG iterations, which yields a significant reduction in runtime. For the 52-investment cases, a larger number of traditional iterations is needed; nevertheless, the number remains substantially lower compared to the traditional approach, resulting in large time savings.

For the End-to-End approach, computational time is reduced in most instances, but the performance is worse compared to the Surrogate method. Furthermore, the End-to-End method took more time compared to the traditional CG for $EPSP\_15\_4$ with 8% growth and $EPSP\_52\_4$ with 8% growth instances. Although the ML iterations can help by decreasing the number of traditional iterations, the time savings are sometimes small. The poorer performance is due to the larger number of traditional CG iterations required to reach the optimality threshold, while only a few ML iterations are executed in some cases.

The reason for the smaller number of ML iterations is an overestimation of the lower bound in the End-to-End scheme. In this approach, the upper bound is computed by solving the RMP with the true operational cost and thus remains valid; however, the lower bound is estimated from the reduced costs associated with the subproblems solved using the investment decision predicted by the End-to-End model. Because the End-to-End model predicts investment decisions without explicitly accounting for the true operational cost, the actual operational cost for a predicted decision can be much larger than anticipated. This may lead to reduced costs that are higher (or even positive) than expected, causing the estimated lower bound to exceed the upper bound and producing an early termination of the ML iterations.

Therefore, the Surrogate-based algorithm constitutes a more effective strategy for predicting negative reduced-cost columns to be added to the RMP. By directly modeling the operational cost, the surrogate approach explicitly accounts for both the investment

and operational components of the reduced cost. This leads to a more accurate estimation of the lower bound, preventing an early termination of ML iterations, yielding to superior computational performance.

Figure 4 illustrates the evolution of the upper and lower bounds obtained by the three column generation approaches—Surrogate-based, End-to-End, and Traditional—for six representative instances: $EPSP_{15\_4\_3}$, $EPSP_{15\_4\_5}$, $EPSP_{15\_12\_3}$, $EPSP_{15\_12\_5}$, $EPSP_{52\_4\_3}$, and $EPSP_{52\_4\_8}$. Dotted lines indicate the iterations performed using ML-enhanced subproblems in the Surrogate and End-to-End approaches, while solid lines correspond to traditional CG iterations.

The surrogate-based approach produces estimated bounds that closely follow the valid bounds, which explains why only a limited number of traditional CG iterations are required after the ML phase. In contrast, the End-to-End approach frequently overestimates the lower bound, leading to premature termination of the ML iterations. As a consequence, additional traditional CG iterations are needed to recover valid bounds and guarantee optimality. This behavior can be attributed to the fact that the End-to-End model does not explicitly account for the operational cost when predicting investment decisions. As a result, columns associated with high operational costs may be generated, yielding reduced costs that are large or even positive. This overestimates the lower bound and limits the effectiveness of the ML-enhanced phase.

Finally, Tables 5 and 6 summarize the computational time and the achieved optimality gap after a single traditional CG iteration and after full convergence to a 0.5% optimality threshold, for the Surrogate-based and End-to-End approaches respectively. For the Surrogate-based method we observe that a single traditional iteration typically yields a low relative optimality gap (often below 5%) within a short runtime, indicating that surrogate iterations successfully generate high-quality columns before performing traditional CG iterations. After full convergence, all instances reach the target gap of 0.5% with moderate additional time in the few cases that required more traditional iterations. The End-to-End approach performed worse for all tested instances; after one traditional iteration the reported gaps are generally larger and, in some instances,

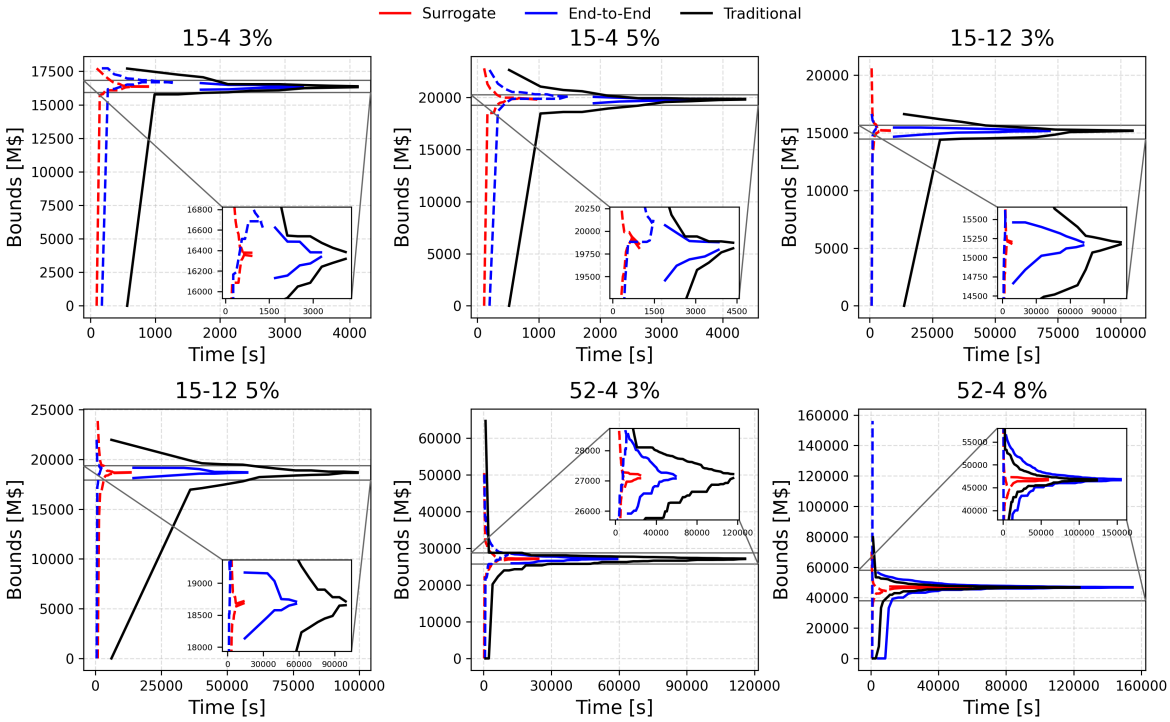


Figure 4: Evolution of upper and lower bounds for six EPSP instances using the Surrogate-based, End-to-End, and Traditional column generation approaches. Dotted lines represent ML-enhanced iterations, while solid lines correspond to traditional CG iterations.

considerably above acceptable levels.

Instance	R_{dem}	After one traditional CG iteration		After convergence	
		Time [hh:mm:ss]	Gap [%]	Time [hh:mm:ss]	Gap [%]
EPSP_15_4	3%	00:14:44	0.186	00:14:44	0.186
EPSP_15_4	5%	00:15:55	0.227	00:15:55	0.227
EPSP_15_4	8%	00:32:18	2.025	00:46:40	0.086
EPSP_15_12	3%	02:12:04	0.161	02:12:04	0.161
EPSP_15_12	5%	03:42:34	0.182	03:42:34	0.182
EPSP_15_12	8%	02:52:05	14.955	04:52:32	0.477
EPSP_52_4	3%	02:38:38	1.734	06:43:22	0.378
EPSP_52_4	5%	02:38:34	2.225	10:18:46	0.494
EPSP_52_4	8%	03:11:43	4.036	16:24:06	0.485

Table 5: Computational time and relative optimality gap obtained after a single traditional iteration of the Surrogate-based column generation algorithm and after convergence to a 0.5% optimality gap.

Instance	R_{dem}	After one traditional CG iteration		After convergence	
		Time [hh:mm:ss]	Gap [%]	Time [hh:mm:ss]	Gap [%]
EPSP_15_4	3%	00:28:23	3.047	00:54:35	0.272
EPSP_15_4	5%	00:31:38	3.122	01:03:52	0.423
EPSP_15_4	8%	00:16:38	70.957	01:22:44	0.316
EPSP_15_12	3%	02:41:53	5.428	19:54:29	0.254
EPSP_15_12	5%	04:01:22	5.685	15:59:12	0.194
EPSP_15_12	8%	01:55:32	$> 100\%$	17:03:27	0.401
EPSP_52_4	3%	03:28:01	10.112	06:43:22	0.298
EPSP_52_4	5%	00:32:45	$> 100\%$	1-06:33:34	0.410
EPSP_52_4	8%	01:11:26	$> 100\%$	1-19:00:31	0.489

Table 6: Computational time and relative optimality gap obtained after a single traditional iteration of the End-to-End-based column generation algorithm and after convergence to a 0.5% optimality gap.

6. Conclusions

In this article, we studied ML-aided column generation methods for solving large-scale capacity planning problems. We consider two ways of approximating or replacing the pricing problem within the column generation framework: (i) a surrogate-based approach, where the operational component is replaced by a pre-trained multi-layer ReLU network embedded via its MILP encoding, and (ii) end-to-end approaches that learn to directly propose new columns. To preserve performance guarantees, our method follows a two-phase procedure: it first solves an approximate optimization problem to quickly generate informative columns, and then switches to standard column generation with the original pricing subproblems to recover valid bounds and compute the optimality gap. Our findings indicate that incorporating an intermediate surrogate phase is worthwhile: it improves computational efficiency, and the columns produced in the approximate phase are sufficiently high-quality to significantly accelerate convergence in the subsequent exact phase.

Acknowledgments

Support was given by Chile/MAGISTER/2025, Fondef ID23I10277. This research was partially supported by the supercomputing infrastructure of the NLHPC (CCSS210001).

Appendix A. Generation Expansion Planning Problem

This section presents the mixed-integer optimization model used for long-term generation expansion planning while explicitly accounting for short-term operational scheduling. The short-term operation is modeled using a standard mixed-integer unit commitment formulation based from [1]. The model determines optimal investment decisions in conventional and renewable generation units, together with hourly unit commitment and dispatch decisions, over a multi-year planning horizon. Investment decisions affect the available generation capacity in future years, whereas operational decisions ensure feasibility with respect to power balance, ramping limits, and network constraints.

The multi-stage power system planning model is presented in (A.1). We consider sets of candidate conventional and renewable generation units, denoted by $\mathcal{G}^i$ and $\mathcal{R}^i$, respectively, which can be installed over the planning years $\mathcal{Y}$. To represent system operation within each year, a set of representative days $\mathcal{D}$ is used, each associated with a weight w_d. Each representative day is modeled with an hourly resolution, with hours indexed by the set $\mathcal{T} = \{1, \ldots, 24\}$.

The objective function (A.1a) minimizes the total system cost, which consists of investment costs and operational costs. Investment costs, account for the installation of new generation units in each year y. New investments, represented by the variables $IG_{y,g}$, are annualized using the corresponding annuity cost $c^{\text{inv}}_{y,g}$ and a discount rate R^{cost}. Operational costs, include the variable generation costs $c^{\text{var}}_{y,g}$ and start-up costs c^{E}_{g} of conventional generators in the set $\mathcal{G}$. In addition, a load-shedding penalty c^{UD}_{b} is introduced at each bus $b \in \mathcal{B}$ to ensure feasibility when demand cannot be fully supplied.

System operation is modeled at an hourly resolution through a set of operational constraints. Constraint (A.1b) enforces power balance at each bus b and hour t, ensuring that the demand $d_{y,b,t}$ is met by the power output of conventional generators $P_{y,g,t}$ and renewable generators $P^{\text{RES}}_{y,r,t}$, net power flows into and out of the bus through transmission lines in the sets $\mathcal{L}^{in}_{b}$ and $\mathcal{L}^{out}_{b}$, and load shedding $LS_{y,b,t}$. Constraint (A.1c) limits the

power flow transmitted over each transmission line $l \in \mathcal{L}$ according to its capacity $\overline{f}_{y,l}$.

The operation of conventional generators is modeled through constraints (A.1d)–(A.1i). Binary variables $u_{y,g,d,t} \in \{0,1\}$ represent the on/off commitment status of each generator at each hour. Variables $X^{\text{ON}}_{y,g,t}$ and $X^{\text{OFF}}_{y,g,t}$ indicate generator start-up and shutdown events, respectively, and are linked to the commitment variables through the unit commitment state equation (A.1e). Constraint (A.1d) enforces the minimum and maximum power output limits of each conventional generator, given by $\underline{p}_g$ and $\overline{p}_g$. Constraints (A.1f) and (A.1g) model ramping limitations by restricting changes in power output between consecutive hours according to ramp-up (Rup_g) and ramp-down (Rdn_g) limits. Finally, constraints (A.1h) and (A.1i) impose minimum up-time τ^{ON}_g and minimum down-time τ^{OFF}_g requirements.

Constraint (A.1l) restricts the commitment of conventional generators to those units that have been previously built. Renewable generation availability is modeled by constraint (A.1m), which limits the renewable power output using the availability factor $\alpha_{r,t}$, the installed renewable capacity $\overline{p}_r$, and the number of renewable units that have been built. Finally, constraints (A.1n)–(A.1r) define the domains of the decision variables, specifying non-negativity constraints for continuous variables and binary restrictions for commitment and investment variables.

$$\min \quad \sum_{y\in\mathcal{Y}} \frac{1}{(1+R^{\text{cost}})^{(y-1)}} \Bigg[\sum_{g\in\mathcal{G}^{\text{conv}}_{\text{new}}} \sum_{l=1}^{y} c^{\text{inv}}_{g} \text{IG}^{\text{conv}}_{g,y} + \sum_{r\in\mathcal{G}^{\text{renw}}_{\text{new}}} \sum_{l=1}^{y} c^{\text{inv}}_{r} \text{IG}^{\text{renw}}_{g,y} + \sum_{d\in\mathcal{D}} \sum_{t\in\mathcal{T}} \left(\sum_{g\in\mathcal{G}} (c^{\text{var}}_{g} P_{g,y,d,t} + c^{\text{ON}}_{g} X^{\text{ON}}_{g,y,d,t}) + \sum_{b\in\mathcal{B}} c^{L}_{b} LL_{b,y,d,t} \right) \Bigg] \tag{A.1a}$$

$$\text{s.t.} \quad \sum_{g\in G_b} P_{g,y,d,t} + \sum_{l\in\mathcal{L}^{in}_{b}} F_{l,y,d,t} - \sum_{l\in\mathcal{L}^{out}_{b}} F_{l,y,d,t} = D_{b,y,d,t} - LL_{b,y,d,t}, \quad \forall y,d,b,t \tag{A.1b}$$

$$-\overline{F}_l \le F_{l,y,d,t} \le \overline{F}_l, \qquad \forall y,d,l,t \tag{A.1c}$$

$$U_{g,y,d,t} \cdot \underline{P}_g \le P_{g,y,d,t} \le \overline{P}_g \cdot U_{g,y,d,t}, \qquad \forall y,d,g,t \tag{A.1d}$$

$$U_{g,y,d,t} - U_{g,y,d,t-1} = X^{\text{ON}}_{g,y,d,t} - X^{\text{OFF}}_{g,y,d,t}, \qquad \forall y,d,g,t \tag{A.1e}$$

$$P_{g,y,d,t} - P_{g,y,d,t-1} \le U_{g,y,d,t} \cdot \overline{R_g} + X^{\text{ON}}_{g,y,d,t} \cdot \underline{P}_g, \qquad \forall y,d,g,t \tag{A.1f}$$

$$P_{g,y,d,t-1} - P_{g,y,d,t} \le U_{g,y,d,t} \cdot \overline{R_g} + X^{\text{OFF}}_{g,y,d,t} \cdot \overline{P}_g, \qquad \forall y,d,g,t \tag{A.1g}$$

$$U_{g,y,d,t} \ge \sum_{t'=t-\tau^{\text{ON}}_{g}}^{t} X^{\text{ON}}_{g,y,d,t'}, \qquad \forall y,d,g,t \tag{A.1h}$$

$$1 - U_{g,y,d,t} \ge \sum_{t'=t-\tau^{\text{OFF}}_{g}}^{t} X^{\text{OFF}}_{g,y,d,t'}, \qquad \forall y,d,g,t \tag{A.1i}$$

$$U_{g,y,d,t} \le \sum_{l=1}^{y} \text{IG}^{\text{conv}}_{g,l}, \qquad \forall y,d,g\in\mathcal{G}^{\text{conv}}_{\text{new}},t \tag{A.1j}$$

$$P_{r,y,d,t} \le \alpha_{r,d,t} \cdot \overline{P}_r, \qquad \forall y,d,r,t \tag{A.1k}$$

$$P_{r,y,d,t} \le \alpha_{r,d,t} \cdot \overline{P}_r \cdot \sum_{l=1}^{y} \text{IG}^{\text{conv}}_{r,l}, \qquad \forall y,d,r\in\mathcal{G}^{\text{renw}}_{\text{new}},t \tag{A.1l}$$

$$U_{g,y,d,t}, X^{\text{ON}}_{g,y,d,t}, X^{\text{OFF}}_{g,y,d,t} \in \{0,1\}, \qquad \forall y,d,g,t \tag{A.1m}$$

$$c^{\text{op}}_{y,d}, c^{\text{inv}}_{y} \ge 0, \qquad \forall y,d \tag{A.1n}$$

$$P_{g,y,d,t} \ge 0, \qquad \forall y,d,g,t \tag{A.1o}$$

$$LL_{b,y,d,t} \ge 0, \qquad \forall y,d,b,t \tag{A.1p}$$

$$\text{IG}^{\text{conv}}_{g,y} \in \{0,1\}, \qquad \forall y,g\in\mathcal{G}^{\text{conv}}_{\text{new}} \tag{A.1q}$$

$$\text{IG}^{\text{renw}}_{g,y} \in \{0,1\}, \qquad \forall y,g\in\mathcal{G}^{\text{renw}}_{\text{new}} \tag{A.1r}$$

Appendix B. Hyperparameter Search

To perform the hyperparameter search, a grid search strategy was adopted. For both ML-based approaches, three neural network architectures were evaluated, represented by the lists [64], [64, 32], and [64, 32, 16], which indicate the number of units in each hidden layer. In addition, five learning rates were tested: 10^{-4}, 5×10^{-4}, 10^{-3}, 5×10^{-3}, and 10^{-2}, together with three dropout probabilities: 0, 0.05, and 0.1.

The hyperparameter search was conducted sequentially. First, the learning rate was selected while keeping the remaining hyperparameters fixed. Then, the network architecture was varied, and finally, the dropout probability was tuned. Tables B.7, B.8, and B.9 report the results of the hyperparameter search for the surrogate-based approach across all instances. Similarly, Tables B.10, B.11, and B.12 present the corresponding results for the End-to-End approach.

Instance	Architecture	Learning Rate	Dropout	MAE
EPSP_15_4	[64, 32]	1e-4	0	180.777
EPSP_15_4	[64, 32]	5e-4	0	33.747
EPSP_15_4	[64, 32]	1e-3	0	17.115
EPSP_15_4	[64, 32]	5e-3	0	11.715
EPSP_15_4	[64, 32]	1e-2	0	14.168
EPSP_15_4	[64]	5e-3	0	14.428
EPSP_15_4	[64, 32, 16]	5e-3	0	12.610
EPSP_15_4	[64, 32]	5e-3	0.05	82.451
EPSP_15_4	[64, 32]	5e-3	0.1	105.983

Table B.7: Hyperparameter search results for the surrogate-based approach on instance EPSP_15_4. The table reports the mean absolute error (MAE) obtained for different combinations of network architectures, learning rates, and dropout probabilities.

Instance	Architecture	Learning Rate	Dropout	MAE
EPSP_15_12	[64, 32]	1e-4	0	157.691
EPSP_15_12	[64, 32]	5e-4	0	67.651
EPSP_15_12	[64, 32]	1e-3	0	18.318
EPSP_15_12	[64, 32]	5e-3	0	16.935
EPSP_15_12	[64, 32]	1e-2	0	12.897
EPSP_15_12	[64]	1e-2	0	21.368
EPSP_15_12	[64, 32, 16]	1e-2	0	16.432
EPSP_15_12	[64, 32]	1e-2	0.05	75.224
EPSP_15_12	[64, 32]	1e-2	0.1	103.930

Table B.8: Hyperparameter search results for the surrogate-based approach on instance EPSP_15_12. The table reports the mean absolute error (MAE) obtained for different combinations of network architectures, learning rates, and dropout probabilities.

Instance	Architecture	Learning Rate	Dropout	MAE
EPSP_52_4	[64, 32]	1e-4	0	788.352
EPSP_52_4	[64, 32]	5e-4	0	658.064
EPSP_52_4	[64, 32]	1e-3	0	348.139
EPSP_52_4	[64, 32]	5e-3	0	133.617
EPSP_52_4	[64, 32]	1e-2	0	160.351
EPSP_52_4	[64]	5e-3	0	494.872
EPSP_52_4	[64, 32, 16]	5e-3	0	151.352
EPSP_52_4	[64, 32]	5e-3	0.05	248.921
EPSP_52_4	[64, 32]	5e-3	0.1	309.949

Table B.9: Hyperparameter search results for the surrogate-based approach on instance EPSP_52_4. The table reports the mean absolute error (MAE) obtained for different combinations of network architectures, learning rates, and dropout probabilities.

Instance	Architecture	Learning Rate	Dropout	BCE
EPSP_15_4	[64, 32]	1e-4	0	0.0729
EPSP_15_4	[64, 32]	5e-4	0	0.0620
EPSP_15_4	[64, 32]	1e-3	0	0.0557
EPSP_15_4	[64, 32]	5e-3	0	0.0754
EPSP_15_4	[64, 32]	1e-2	0	0.1258
EPSP_15_4	[64]	1e-3	0	0.0585
EPSP_15_4	[64, 32, 16]	1e-3	0	0.0534
EPSP_15_4	[64, 32, 16]	1e-3	0.05	0.0872
EPSP_15_4	[64, 32, 16]	1e-3	0.1	0.1079

Table B.10: Hyperparameter search results for the End-to-End-based approach on instance EPSP_15_4. The table reports the Binary Cross-Entropy with Logits loss (BCE-with-logits) obtained for different combinations of network architectures, learning rates, and dropout probabilities.

Instance	Architecture	Learning Rate	Dropout	BCE
EPSP_15_4	[64, 32]	1e-4	0	0.0624
EPSP_15_4	[64, 32]	5e-4	0	0.0482
EPSP_15_4	[64, 32]	1e-3	0	0.0436
EPSP_15_4	[64, 32]	5e-3	0	0.0653
EPSP_15_4	[64, 32]	1e-2	0	0.1090
EPSP_15_4	[64]	1e-3	0	0.0474
EPSP_15_4	[64, 32, 16]	1e-3	0	0.0459
EPSP_15_4	[64, 32]	1e-3	0.05	0.0531
EPSP_15_4	[64, 32]	1e-3	0.1	0.0649

Table B.11: Hyperparameter search results for the End-to-End-based approach on instance EPSP_15_12. The table reports the Binary Cross-Entropy with Logits loss (BCE-with-logits) obtained for different combinations of network architectures, learning rates, and dropout probabilities.

Instance	Architecture	Learning Rate	Dropout	BCE
EPSP_15_4	[64, 32]	1e-4	0	0.1144
EPSP_15_4	[64, 32]	5e-4	0	0.1386
EPSP_15_4	[64, 32]	1e-3	0	0.1712
EPSP_15_4	[64, 32]	5e-3	0	0.2549
EPSP_15_4	[64, 32]	1e-2	0	0.2698
EPSP_15_4	[64]	1e-4	0	0.0916
EPSP_15_4	[64, 32, 16]	1e-4	0	0.1729
EPSP_15_4	[64]	1e-4	0.05	0.1450
EPSP_15_4	[64]	1e-4	0.1	0.1615

Table B.12: Hyperparameter search results for the End-to-End-based approach on instance EPSP_52_4. The table reports the Binary Cross-Entropy with Logits loss (BCE-with-logits) obtained for different combinations of network architectures, learning rates, and dropout probabilities.

References

[1] A. Flores-Quiroz, K. Strunz, A distributed computing framework for multi-stage stochastic planning of renewable power systems with energy storage as flexibility option, Applied Energy 291 (2021) 116736.

[2] J. Ma, V. Silva, R. Belhomme, D. S. Kirschen, L. F. Ochoa, Evaluating and Planning Flexibility in Sustainable Power Systems, IEEE Transactions on Sustainable Energy 4 (1) (2013) 200–209. doi:10.1109/TSTE.2012.2212471.
URL http://ieeexplore.ieee.org/document/6313967/

[3] A. F. Abdin, E. Zio, An integrated framework for operational flexibility assessment in multi-period power system planning with renewable energy production, Applied Energy 222 (2018) 898–914. doi:10.1016/j.apenergy.2018.04.009.
URL https://linkinghub.elsevier.com/retrieve/pii/S0306261918305518

[4] Z. Tian, X. Li, J. Niu, R. Zhou, F. Li, Enhancing operation flexibility of distributed energy systems: A flexible multi-objective optimization planning method considering long-term and temporary objectives, Energy 288 (2024) 129612. doi:10.1016/j.energy.2023.129612.
URL https://linkinghub.elsevier.com/retrieve/pii/S0360544223030062

[5] T. Rathi, B. P. Riley, A. Flores-Quiroz, Q. Zhang, Column generation for multistage stochastic mixed-integer nonlinear programs with discrete state variables, Journal of Global Optimization (2025). doi:10.1007/s10898-025-01480-x.
URL https://doi.org/10.1007/s10898-025-01480-x

[6] A. Flores-Quiroz, J. M. Pinto, Q. Zhang, A column generation approach to multiscale capacity planning for power-intensive process networks, Optimization and Engineering 20 (4) (2019) 1001–1027. doi:10.1007/s11081-019-09435-4.
URL https://doi.org/10.1007/s11081-019-09435-4

[7] K. J. Singh, A. B. Philpott, R. K. Wood, Dantzig-wolfe decomposition for solving multistage stochastic capacity-planning problems, Operations Research 57 (5) (2009) 1271–1286.

[8] C. Saldarriaga-Cortés, H. Salazar, R. Moreno, G. Jiménez-Estévez, Stochastic planning of electricity and gas networks: An asynchronous column generation approach, Applied energy 233 (2019) 1065–1077.

[9] P. Apablaza, S. Püschel-Løvengreen, R. Moreno, S. Mhanna, P. Mancarella, Assessing the impact of der on the expansion of low-carbon power systems under deep uncertainty, Electric Power Systems Research 235 (2024) 110824.

[10] M. E. Lübbecke, J. Desrosiers, Selected Topics in Column Generation, Operations Research 53 (6) (2005) 1007–1023. doi:10.1287/opre.1050.0234.
URL https://pubsonline.informs.org/doi/10.1287/opre.1050.0234

[11] A. Hijazi, O. Ozaltin, R. Uzsoy, All you need is an improving column: Enhancing column generation for parallel machine scheduling via transformers, version Number: 1. doi:10.48550/ARXIV.2410.15601.
URL https://arxiv.org/abs/2410.15601

[12] Y. Liu, R. Sioshansi, A. J. Conejo, Multistage stochastic investment planning

with multiscale representation of uncertainties and decisions, IEEE Transactions on Power Systems 33 (1) (2017) 781–791.

[13] F. D. Munoz, J.-P. Watson, A scalable solution framework for stochastic transmission and generation planning problems, Computational Management Science 12 (4) (2015) 491–518.

[14] P. Falugi, I. Konstantelos, G. Strbac, Planning with multiple transmission and storage investment options under uncertainty: A nested decomposition approach, IEEE Transactions on Power Systems 33 (4) (2017) 3559–3572.

[15] C. Li, A. J. Conejo, P. Liu, B. P. Omell, J. D. Siirola, I. E. Grossmann, Mixed-integer linear programming models and algorithms for generation and transmission expansion planning of power systems, European Journal of Operational Research 297 (3) (2022) 1071–1082.

[16] C. L. Lara, J. D. Siirola, I. E. Grossmann, Electric power infrastructure planning under uncertainty: stochastic dual dynamic integer programming (sddip) and parallelization scheme, Optimization and Engineering 21 (4) (2020) 1243–1281.

[17] J. Zou, S. Ahmed, X. A. Sun, Stochastic dual dynamic integer programming, Mathematical Programming 175 (1) (2019) 461–502.

[18] C. Chi, A. Aboussalah, E. Khalil, J. Wang, Z. Sherkat-Masoumi, A deep reinforcement learning framework for column generation, Advances in Neural Information Processing Systems 35 (2022) 9633–9644.

[19] Y. Shen, Y. Sun, X. Li, A. Eberhard, A. Ernst, Enhancing column generation by a machine-learning-based pricing heuristic for graph coloring, in: Proceedings of the AAAI conference on artificial intelligence, Vol. 36, 2022, pp. 9926–9934.

[20] P. Sarkar, V. B. Khanapuri, M. K. Tiwari, Accelerating the stabilized column generation using machine learning, Computers & Industrial Engineering 200 (2025) 110837.

[21] S. Kraul, M. Seizinger, J. O. Brunner, Machine learning–supported prediction of dual variables for the cutting stock problem with an application in stabilized column generation, INFORMS Journal on Computing 35 (3) (2023) 692–709.

[22] N. Sugishita, A. Grothey, K. McKinnon, Use of machine learning models to warm-start column generation for unit commitment, INFORMS Journal on Computing 36 (4) (2024) 1129–1146.

[23] F. Quesnel, A. Wu, G. Desaulniers, F. Soumis, Deep-learning-based partial pricing in a branch-and-price algorithm for personalized crew rostering, Computers & Operations Research 138 (2022) 105554.

[24] J. Gerbaux, G. Desaulniers, Q. Cappart, A machine-learning-based column generation heuristic for electric bus scheduling, Computers & Operations Research 173 (2025) 106848.

[25] K. Xu, L. Shen, L. Liu, Enhancing column generation by reinforcement learning-based hyper-heuristic for vehicle routing and scheduling problems, Computers & Industrial Engineering (2025) 111138.

[26] F. Vanderbeck, Implementing mixed integer column generation, in: G. Desaulniers, J. Desrosiers, M. M. Solomon (Eds.), Column Generation, Springer-Verlag, pp. 331–358. doi:10.1007/0-387-25486-2_12.
URL http://link.springer.com/10.1007/0-387-25486-2_12

[27] J. Dumouchelle, R. Patel, E. B. Khalil, M. Bodur, Neur2SP: Neural Two-Stage Stochastic ProgrammingArXiv:2205.12006 [cs, math] (Oct. 2022). doi:10.48550/arXiv.2205.12006.
URL http://arxiv.org/abs/2205.12006

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

[29] M. L. Bynum, G. A. Hackebeil, W. E. Hart, C. D. Laird, B. L. Nicholson, J. D. Siirola, J.-P. Watson, D. L. Woodruff, Pyomo–optimization modeling in python, 3rd Edition, Vol. 67, Springer Science & Business Media, 2021.

[30] Gurobi Optimization, LLC, Gurobi Optimizer Reference Manual (2023).
URL https://www.gurobi.com

[31] 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, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[33] C. E. Nacional, Coordinador eléctrico nacional (cen), official electricity system data for the Chilean power system (2025).
URL https://www.coordinador.cl/

[34] I. J. Scott, P. M. Carvalho, A. Botterud, C. A. Silva, Clustering representative days for power systems generation expansion planning: Capturing the effects of variable renewables and energy storage, Applied Energy 253 (2019) 113603. doi:10.1016/j.apenergy.2019.113603.
URL https://linkinghub.elsevier.com/retrieve/pii/S0306261919312772