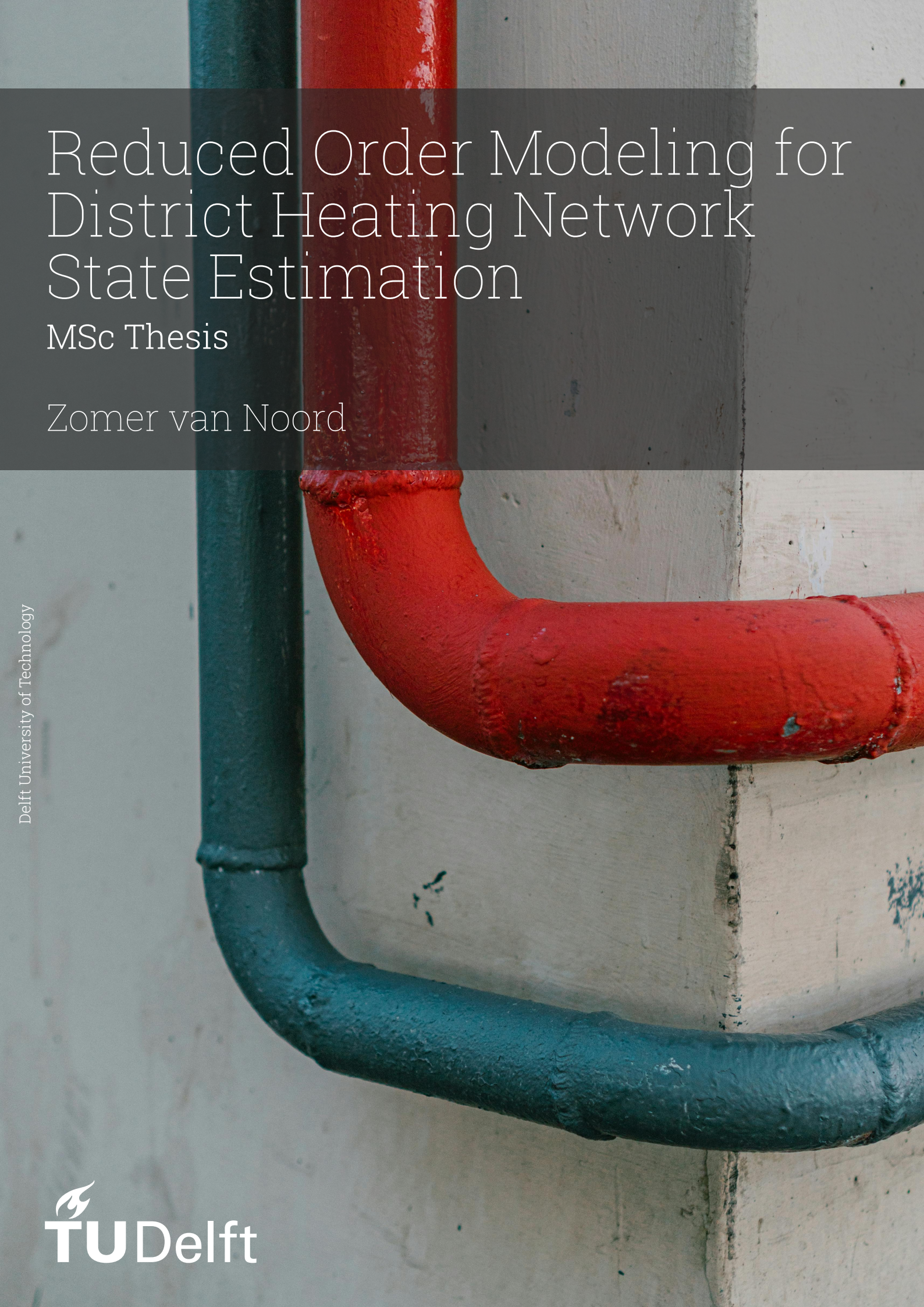


Reduced Order Modeling for District Heating Network State Estimation

MSc Thesis

Zomer van Noord

Delft University of Technology

TU Delft

Reduced Order Modeling for District Heating Network State Estimation

MSc Thesis

by

Zomer van Noord

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday the 24th of March, 2026

Student number:	5125049	
Project duration:	July 1, 2025 – March 24, 2026	
Thesis committee:	Dr. A. Heinlein,	TU Delft, supervisor
	M. Kemna,	Gradyent, supervisor
	Prof. dr. H. Schuttelaars,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

The real-time optimization of District Heating (DH) networks is computationally demanding due to the strong interdependence of their state variables. This thesis investigates whether a Reduced Order Model (ROM) can improve the computational efficiency of state estimation within DH networks. Existing ROM approaches either rely on simplified physical assumptions or require manual, topology-dependent adjustments to the model architecture, which prevents fully automated retraining. In contrast, the method developed in this work is based solely on Full Order Model (FOM) data and follows a uniform training pipeline that can be applied to any DH network without human analysis of its topology or case-specific architectural modifications. A hybrid Proper Orthogonal Decomposition (POD) framework is introduced, in which POD extracts dominant spatial modes from the high-dimensional FOM data, while a feedforward neural network predicts the corresponding temporal coefficients from compressed input features. The ROM output is subsequently used as the initial guess in the FOM's state iteration procedure, ensuring that physical consistency is preserved. The approach is evaluated on two realistic DH networks of different scales. In both cases, the ROM achieves total relative reconstruction errors below 5% (specifically 4.8% for the smaller network and 3.6% for the larger network), with prediction times under 0.1 seconds, a substantial improvement compared to the roughly 100 seconds required for a single FOM iteration. For the smaller network, integrating the ROM into the optimization workflow results in a 1.17$\times$ speed-up while yielding decision variables nearly identical to those of the FOM. The resulting speed-up is driven by skipping the first FOM iteration, reducing the total number of iterations needed for convergence, and requiring updates for fewer time steps per iteration. For the larger network, the ROM maintains high predictive accuracy but does not perform reliably during optimization, possibly due to the limited availability of training data for rarely activated back-up sources. Overall, the results demonstrate that hybrid POD ROMs can offer substantial computational savings for DH state estimation and optimization, though their reliability ultimately depends on whether the training dataset adequately captures the operational regimes of all sources.

Declaration of the use of AI

I hereby declare that I have used generative AI solely to improve the clarity, structure, and quality of my own writing. Generative AI was not used to create text or content on my behalf.

Contents

1

Introduction

Aligned with the European Union's climate targets of a 55% reduction in greenhouse gas emissions by 2030 (relative to 1990 levels) and climate neutrality by 2050, reducing CO_2 emissions has become a priority [13]. While much attention has been given to the electrification of transportation and the phase-out of fossil fuels such as gas and oil, one critical area often overlooked is heating. In 2020, EU buildings used 42% of all energy delivered for direct consumption. Households accounted for two-thirds of this share [14]. Optimizing and making heating sources more efficient results in lower emissions and more responsible resource use.

District Heating (DH) is an energy distribution system that delivers heat generated at one or more centralized facilities to multiple buildings within a defined area via a network of insulated pipes. Heated water is transported through a supply network to individual buildings, where heat exchangers transfer the thermal energy for space and water heating. The cooled water is then returned to the central plant via a return network for reheating. This centralized approach results in higher energy efficiency and improved pollution control because DH systems can use renewable sources, waste heat, or efficient fossil fuel plants for heat generation and distribute it through well-insulated networks. In contrast, decentralized systems like individual gas boilers or electric heaters typically have higher emissions per unit of heat and less efficient energy use, leading to larger carbon footprints [52].

In Europe, there are approximately 6,000 heating and cooling networks that supply thermal energy to approximately 100 million people across 32 countries. However, the distribution of these systems varies significantly across countries. For instance, in Denmark, about 64% of households are connected to a DH network, with coverage reaching nearly 98% in the capital, Copenhagen. In contrast, the Netherlands has a much lower adoption rate, with fewer than 5% of households currently connected to such systems [46].

A digital twin is a virtual representation of a physical system, which operates in real-time by continuously updating as new data becomes available. Gradyent, a company providing optimization solutions for heating and cooling networks, applies this concept to DH networks across more than 35 European cities [21]. Their digital twin is a virtual copy of a complete heating (or steam) system and integrates sensor, weather, and demand data to enable immediate operational adjustments. By incorporating weather and demand forecasts, the digital twin also optimizes future production planning and adapts to changing conditions. This approach leads to a more sustainable and efficient use of heating sources and minimizes emissions. Gradyent's projects implemented in various European cities have demonstrated that the application of the digital twin results in 15% reduction in heat loss and a 10% decrease in CO_2 emissions [21].

Gradyent's model is tasked with determining the optimal distribution of limited heat across the network to meet demand constraints. Mathematically, this is formulated as a nonlinear optimization problem in which the decision variables are the supply temperatures of the heat sources. The objective minimizes total heat production while enforcing hydraulic and thermal constraints derived from the underlying network physics. These constraints depend on the state of each component in the network: pipes, sub-

stations, sources, pumps, valves, and shunts. The state comprises the pressure, flow rate, power, and temperature associated with each component, all of which affect how heat can be efficiently allocated. The *network state estimator* computes these variables using hydraulic and heat transfer principles on demand and heat source data. An explanation of this solver is provided in section 2.3.

These state variables are interdependent in two ways: within each component and across the network. Inside a single component, changes in one variable affect others (e.g. a higher flow rate increases the pressure drop across a pipe). Across the network, components influence each other as well; the temperature at one pipe segment depends on the flow and temperature conditions of the upstream segments, which in turn affect downstream temperatures. To capture these interactions, the network state estimator uses an iterative process that refines estimates until all variables stabilize. Its runtime depends on both network size and simulation horizon. For large networks ($\approx$400 thermal loads), a state estimator with a 24-hour horizon can take up to an hour, and the optimization can require up to two hours on the hardware described in section 6.1.

The optimization process repeatedly uses the network state estimator to evaluate possible configurations of decision variables and selects the optimal one. This repeated use makes the current optimization computationally expensive. To address this, a surrogate model is proposed to approximate the network state estimator, which in this work represents the Full Order Model (FOM). The predictions of the surrogate model are implemented as an initial guess in the fixed-point iteration of the FOM, aiming to deliver estimates in a shorter amount of computation time while maintaining acceptable accuracy. To preserve the essence and functionality of the network state estimator, the surrogate model will be implemented as a *Reduced Order Model* (ROM). A ROM is a simplified approximation of the FOM, achieved by reducing the state-space dimension, decreasing the degrees of freedom, or constructing a fully data-based input-output approximation [56].

Applying ROM techniques to accelerate numerical simulations or optimal control computations of DH networks has been explored in previous studies [5, 19, 22, 28, 51, 53]. Existing approaches can be broadly categorized into intrusive and non-intrusive methods [45]. Intrusive methods, such as those proposed by Rein [51], directly manipulate the governing equations of the FOM. His approach achieved substantial computational speed-up but relies on strong simplifying assumptions, such as neglecting heat losses and assuming uniform pipe geometry, which makes them not applicable to realistic DH networks.

Non-intrusive methods, on the other hand, leverage simulation data rather than governing equations, but most existing approaches remain tightly coupled to the specific DH network they are trained on. This approach is preferred when the underlying physics is unknown or too complex to be expressed in a closed set of equations. For example, Boussaid et al. [5] employed graph neural networks to predict temperatures, while Dubon et al. [53] combined network clustering (i.e. grouping similar substations into representative clusters) with neural networks to estimate outgoing temperatures for clustered substations. Similarly, De Giuli et al. [19] embedded the DH network topology into a neural network architecture. Although these methods achieved significant speed-ups, they share key limitations: their architectures depend on the specific network topology, and the number or structure of the neural networks must be adapted whenever the number of pipes, substations, or clusters changes [5, 19, 53]. As a result, this requires human analysis of each new network before training can begin, making these approaches difficult to reuse in a uniform way across different DH systems.

This gap underscores the need for generalizable Model Order Reduction (MOR) strategies that can calculate state variables without requiring network-specific architectural decisions or manual preprocessing, while still remaining compatible with the complexity of real-world DH systems. To address this, the present work proposes a hybrid approach that combines intrusive and data-driven techniques. Specifically, Proper Orthogonal Decomposition (POD) is applied to compress the FOM state into dominant spatial modes and associated temporal coefficients. A feedforward neural network then predicts future values of these coefficients from the same input conditions used by the network state estimator. The entire pipeline can be applied to a new DH network simply by providing new FOM simulation data, without requiring any human intervention to redesign or adapt the model architecture.

Hybrid POD has demonstrated substantial computational speed-up in other domains [26, 55, 59, 63, 65]. The choice of a non-intrusive approach is motivated by the existence of the iterative process within

the network state estimator, which lacks explicit governing equations. Furthermore, applying POD prior to training reduces dimensionality and provides an intermediate error, allowing deviations to be traced back to specific steps in the pipeline rather than having a fully black-box approach.

This thesis is structured as follows. Chapter 2 describes Gradyent's digital twin, with an emphasis on the network state estimator solver and the optimization method. Chapter 3 gives background on surrogate models and ROMs, distinguishing between intrusive and non-intrusive approaches. It includes a more detailed explanation of POD and the fundamentals of neural networks. It concludes by introducing performance metrics for evaluating ROMs. Chapter 4 reviews prior research on MOR applied to DH networks, as well as highlighting hybrid POD approaches to other applications. Chapter 5 gives details on the methodology and the motivation for our approach. Chapters 6 and 7 contain the results of the constructed ROM applied to a small and large DH test case, respectively. Finally, this thesis concludes with a summary of the findings, a discussion and potential further research directions in chapter 8.

This thesis addresses the following main research question:

> Can a ROM enhance computational efficiency in calculating state variables within a DH network?

To answer this question, three subquestions are formulated:

1. *How can ROM methods be combined with neural networks to construct a ROM that can be re-trained for different DH networks without manual adaptation of the model architecture?*
2. *How accurately does the ROM predict the FOM's state variables, and how much faster can it computes these states for realistic DH networks?*
3. *Can using the ROM to generate an initial guess in the optimization yield a speed-up in optimization runtime for such networks?*

The first subquestion is addressed in chapter 5, while the other two are addressed in chapters 6 and 7. The conclusion to the main research question follows in chapter 8.

2

Modeling a District Heating (DH) network

This chapter introduces a framework for modeling DH networks and details Gradyent's implementation. It starts with an overview of Gradyent's digital twin platform in section 2.1, followed by a discussion of the main components of a DH network in section 2.2. The computation of variable values through Gradyent's network state estimator solver is explained in section 2.3. Finally, section 2.4 provides further insights into the optimization method.

2.1. The digital twin

Gradyent's real-time digital twin is an online platform that provides clients with a dynamic digital copy of their DH network. By integrating customer consumption patterns with geographical, weather and sensor data, the digital twin continuously optimizes control variables of the grid in real-time. The *network state estimator* solver (see section 2.3) computes the state of each network component (pipes, substations, sources, pumps, valves, and shunts). The state refers to the set of physical and control variables that describe the current condition of the system. These values are derived by applying physical laws and conservation principles. This process provides insights for the entire network, even for locations without sensors or smart meters.

Once the network state is computed, control variables are optimized to minimize heat production while ensuring that all demand is satisfied. This stage, the data estimation and optimization, is particularly relevant to our work, as it represents the most computationally intensive part of the model. Our focus is on implementing a ROM for the network state estimator solver, with the goal of accelerating the optimization process and significantly reducing computation time.

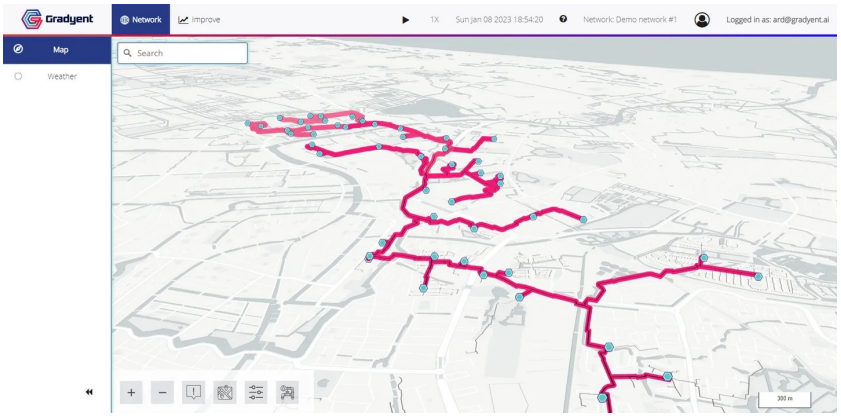


Figure 2.1: Example of the digital twin platform of a DH network [21]

2.2. Model components

Figure 2.2 illustrates an example of a DH network and its main components. In a DH network, one or more heat *sources* raise the temperature of cooled water to the desired level and inject it into the *forward network*. This section of the system transports hot water from the thermal energy sources to buildings with specific heat demands, referred to as *thermal loads* or *sinks*. Inside these buildings, heat exchangers extract thermal energy from the hot water for internal heating and return the cooled water to the *return network*. Finally, the return network transports the cooled water back to the heating sources for reheating. Together, the forward and return networks form a closed-loop system for water circulation. While the number of sources and thermal loads may vary, each supply pipe in the forward network is typically paired with a corresponding pipe in the return network.

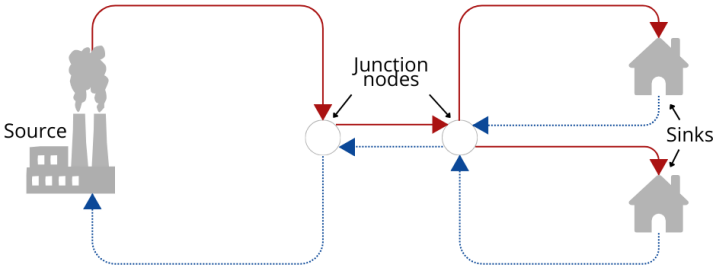


Figure 2.2: Schematic representation of a DH network. Forward pipes are shown in solid red, return pipes in dashed blue.

In addition, DH networks incorporate pumps, valves and shunts to regulate the flow rates locally. Each heat source is equipped with a pump that drives the flow into the forward network. At the thermal loads, valves adjust the water flow through the heat exchanger: opening slightly to increase flow or closing to reduce it. By increasing the local flow rate, a heat exchanger at a sink can extract a larger amount of thermal energy from the network. Valves are usually partially closed by default and gradually open wider when additional heat is required. Some DH networks also include pumps and valves within the network to regulate the flow rates locally at multiple places. In a DH network, shunts blend a portion of the cooler return water with the hot supply water to lower and precisely regulate the supply temperature. This is especially valuable in larger networks where different areas require different temperature levels. By mixing in cooler water, the supply temperature can be reduced, which in turn decreases heat losses.

In the digital twin of the DH network, individual buildings are not always represented as separate thermal loads. Instead, buildings with similar demand profiles can be combined into a single aggregated load, a process known as *clustering*.

The key variables in DH network modeling are temperature T ($^{\circ}C$), pressure p (bar), flow rate Q (m^3/s) and thermal power P (W). Measuring temperature is critical because thermal loads require water at specific temperatures. The pressure differences between sources and loads is what drives the water flow through the network. Moreover, water needs to be maintained at a sufficiently high pressure to ensure it remains in the liquid phase, as the heat exchangers in DH networks operate only with liquid water[51]. As previously discussed, measuring and regulating the flow rate enables precise control over the water temperature. Thermal power is measured to quantify the heating demand or production at a single node. Typically, water reaches the thermal loads at approximately $70\,^{\circ}\mathrm{C}$ to $130\,^{\circ}\mathrm{C}$ and returns at about $30\,^{\circ}\mathrm{C}$ to $80\,^{\circ}\mathrm{C}$. The system pressures can reach up to 16 bar [27].

As water moves through pipes, it experiences friction against the pipe walls, resulting in a pressure drop. This drop depends on pipe length and diameter, fluid density, flow velocity, and the friction factor [7]. This friction factor is a dimensionless number that represents the resistance to flow caused by the pipe's internal surface roughness and the fluid's flow regime (laminar or turbulent). Older, rougher pipes increase friction and cause pressure losses. Higher flow rates further raise friction, and together these factors lead to a pressure drop.

Furthermore, heat loss during water transport through the heating network is another key factor which needs to be taken into account. Losses depend on the temperature difference between the pipe and surrounding soil, the insulation's thermal conductivity, and pipe length [61]. Predictions of the ambient

temperature are therefore incorporated into the model. Heat loss is calculated per pipe segment: the outlet temperature equals the inlet temperature minus the computed loss. Longer pipes generally lose more heat, and lower flow rates indirectly affect the heat losses as lower flow rates result in water remaining in the pipe longer. Conversely, higher source temperatures result in hotter water throughout the transportation but an increase in heat loss due to a larger temperature gradient with the soil [62]. Reducing source temperature thus mitigates losses but risks insufficient heating for distant consumers. The model determines this optimal trade-off.

2.3. Network state estimator

Client-provided data generally includes the demand profiles of the sinks and the power output of the thermal energy sources. What is absent, however, is information about the key variables at the pipes, sources and sinks within the DH network. To estimate these missing variables, the network state estimator solver is employed.

To describe this process, we define a mapping $F : U \to S$, where U denotes the space of boundary conditions, and S denotes the space of network states. The mapping F represents the complete set of physical relations and network-specific constraints. This includes the interactions between flows, pressures, and temperatures within each component; the operational behavior of pumps, valves, shunts, and thermal loads; and the boundary conditions imposed by source settings and demands. Given boundary conditions $u \in U$, the mapping F returns a network state $x = F(u)$ that satisfies these relationships and constraints. For confidentiality reasons, no details can be provided regarding the specific physical equations, numerical solution methods, solver inputs, or internal graph representation used in the estimator.

A single evaluation of F is not sufficient to obtain a fully consistent state, because DH networks exhibit strong interdependence between variables. Not only do variables within a single component influence each other (e.g., higher flow rate implies higher pressure difference between the start and end of a pipe), but thermal variables also propagate across neighboring components. For instance, the temperature at one pipe segment depends on the flow and temperature conditions of the upstream segments, which in turn affect downstream temperatures. Consequently, an iterative approach is required to adjust these estimates until all variables stabilize and demand constraints are satisfied. This procedure is referred to as the *state iteration* and can be formulated as a fixed-point iteration.

To describe this iteration, we introduce a mapping $G : S \to S$, which applies F once to update the current estimate of the network state. The fixed-point iteration begins with an initial guess $x^{(0)}$. At each step, the mapping G takes the current estimate $x^{(k)}$ and returns an updated state

$$x^{(k+1)} = G(x^{(k)}).$$

The iterative process is therefore given by repeated application of G until convergence, which is reached when the difference between successive states satisfies

$$||x^{(k+1)} - x^{(k)}|| \leq \epsilon.$$

Once this criterion is met, the resulting state is considered stable and consistent with all network constraints.

This iterative procedure is the main contributor to the computational cost of data estimation. For a large network (3 sources and $\approx$ 400 sinks), the cost per iteration can reach up to 7 minutes for a 24-hour time horizon using the hardware described in section 6.1.

2.4. Optimization

To determine the most efficient distribution of heat within the network, an optimization process is performed. The decision variables in this process are the forward temperatures at the heat sources. Adjusting these variables influences the behavior of other components in the system, including internal network components and other source-related parameters. The objective function aims to minimize the source temperature, directly reducing the energy required for heat generation. For confidentiality reasons, no details can be provided regarding the specific constraints subject to the optimization problem.

The optimization algorithm iteratively moves from one candidate solution to the next. Each candidate represents a full configuration across all time steps, meaning it contains a proposed state for the entire optimization period. The number of candidates evaluated is fixed in advance and depends on the project. Starting from an initial candidate, the algorithm proposes new ones by taking steps in a chosen search direction, with the step size gradually decreasing over time.

Candidate updates are guided by how much flexibility remains to lower the forward temperatures at the sources. This flexibility is determined by the gap between the temperatures predicted by the state estimator and the minimum temperatures required to satisfy hydraulic and thermal constraints at the loads. If a candidate's temperatures approach these limits, the room for further reduction becomes smaller. For every candidate, the network state estimator computes all state variables for each time step, after which the algorithm updates the best configuration per time step based on the candidate's cost and any constraint violations.

Convergence for a candidate is evaluated independently at each time step. Although the optimization problem itself is time-independent, the candidate specifies a full set of states across the optimization horizon, and each time step is optimized as a separate problem. A candidate is declared converged once all of its time step have individually converged. The convergence criteria depend on the specific test case and will be discussed in section 6.7. The whole optimization process terminates when all candidates have reached convergence.

A notable characteristic of this approach is that not every iteration of the network state estimator updates all time steps. Each time step is evaluated independently, and it is only updated when the new candidate provides a better feasible state than the current best one for that time step. If the candidate does not improve a particular time step, that time step remains unchanged. As a result, an iteration may update many time steps or only a single one. In some cases, the search may effectively "stall" on a particular time step that repeatedly fails to find an improved state, even while the other time steps have already converged.

In this work, a steady-state formulation of the optimization problem is studied, in which each time step is solved independently without information from previous ones. To ensure physically consistent behavior across time, additional constraints are imposed that limit the maximum allowable change in flow or temperature between consecutive time steps. These constraints prevent unrealistic jumps and enforce physical realistic solutions.

In summary, the algorithm proceeds as follows:

Algorithm 1 Optimization procedure

```
Initialize candidate c^(0) (set of source temperatures for all time steps)
for k = 0 to k_max do                                          ▷ Loop over candidates
    Generate updated candidate c^(k+1) using a decreasing step size
    for each time step t do
        Estimate network state x_t using the state estimator
        Check convergence of time step t
        if x_t improves the best known state at time t then
            Update best state for time step t
        end if
    end for
    if all time steps have converged then
        break
    end if
end for
```

Due to the repeated use of the network state estimator for each candidate and time step, this approach is computationally intensive. Optimization over a span of 24 hours of a network with 3 sources and 397 sinks can reach up to almost 2 hours using the hardware described in section 6.1.

3

Reduced Order Model (ROM) methods

This chapter outlines the key mathematical concepts underlying the ROM developed in this work and its comparison to existing literature. It begins with an overview of surrogate models and their distinction from ROMs in section 3.1, followed by a discussion of intrusive versus non-intrusive reduction methods in section 3.2. Section 3.3 provides an explanation of the POD method. The fundamentals of feedforward neural networks are discussed in section 3.4. Finally, section 3.5 discusses performance metrics used to evaluate ROMs in existing literature and in this work.

3.1. Surrogate models

Surrogate models approximate the input-output mapping of complex systems. The objective is to achieve significant computational speed-ups while preserving an acceptable level of accuracy [30]. Analyzing the lost accuracy is important, as small prediction errors can propagate and lead to significant inaccuracies during optimization or decision-making. Another common application of surrogate modeling is in uncertainty quantification, where the goal is to assess the reliability of model predictions under varying input conditions [4].

Depending on the required level of accuracy, surrogate models may either fully replace the original high-fidelity model or serve as an initial approximation, with results subsequently verified using the FOM [64]. Another strategy involves integrating surrogate models into a multi-fidelity framework. One common strategy is to use the surrogate only within a region where its accuracy is deemed sufficient, while the FOM is used otherwise. Incorporating real-world data into this process enables continuous improvement of the surrogate model. This data-integration and switching mechanism can be automated to optimize performance and reliability [33, 64].

A ROM is a reduced representation of the FOM, achieved by reducing the state-space dimension, decreasing the degrees of freedom, or constructing a fully data-based input-output approximation [56]. When used as a substitute for the FOM in fast evaluations, a ROM effectively serves as a surrogate. ROMs may be derived from governing physical equations or constructed entirely through data-driven approaches; however, they are typically linked to solution snapshots of the FOM to preserve some degree of physical consistency. In contrast, surrogate models are generally purely data-driven and focus on reproducing input-output relationships rather than the full system state [17].

3.2. Intrusive vs. non-intrusive ROMs

The concept of ROMs originated in systems and control theory, aiming to simplify dynamical systems while preserving their input-output behavior [56]. Since their introduction, ROMs have been widely adopted in numerical analysis. Initially based on classical mathematical techniques, the field has evolved to include data-driven and machine learning-based approaches. This evolution introduces an important distinction when classifying ROM methods: whether they are *intrusive* or *non-intrusive* [45].

Intrusive methods directly manipulate the governing equations of the FOM. Access to the FOM is there-

fore essential, and the resulting ROM strongly relies on the underlying physics of the system. Popular intrusive methods are often projection-based, where the original system of equations is projected onto a lower-dimensional subspace that captures the most representative modes of the data [45].

Non-intrusive methods, on the other hand, rely solely on simulation data. These data-driven approaches operate like a “black box,” learning system behavior directly from data rather than from physical laws. This is particularly useful when the governing equations are unknown or the system is too complex to be accurately described by a closed set of equations [45]. A widely used non-intrusive technique is Dynamic Mode Decomposition (DMD) (see section A.1), which approximates the evolution of a system by identifying a linear operator that best predicts the progression of snapshots in time [34].

The choice between intrusive and non-intrusive ROMs depends primarily on access to the FOM, but also on secondary priorities such as preserving physical fidelity, ensuring numerical stability, or achieving high accuracy. Various performance metrics are discussed in section 3.5. Furthermore, the motivation for the specific choice made in this work is presented in section 5.1.

3.3. Proper Orthogonal Decomposition (POD)

POD is a dimensionality reduction method that approximates high-dimensional system states using a small number of orthonormal basis vectors [56]. The basis matrix is defined as

$$\Phi = [\varphi_1, ..., \varphi_r], \Phi \in \mathbb{R}^{n \times r},$$

where each basis vector $\varphi_i \in \mathbb{R}^n$ is derived from the *snapshot matrix*.

A *snapshot* is a column vector representing the state of a system at a specific time, obtained from simulations or experiments. The snapshots $x(t_1), x(t_2), ..., x(t_m) \in \mathbb{R}^n$ are assembled in the snapshot matrix

$$X = [x(t_1), x(t_2), ..., x(t_m)] \in \mathbb{R}^{n \times m}.$$

The correlation matrix $C \in \mathbb{R}^{n \times n}$ is defined as

$$C := \frac{1}{m} X X^T = \frac{1}{m} \sum_{i=1}^{m} x_i x_i^T.$$

Since C is symmetric and positive semi-definite, its eigenvalues can be ordered as

$$\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_m \geq 0.$$

The reduced basis $\{\varphi_j\}_{j=1}^r$ consists of the r eigenvectors corresponding to the r largest eigenvalues.

In essence, the snapshot matrix is approximated by

$$X \approx \sum_{i=1}^{r} \alpha_i(t) \varphi_i$$

where $\{\varphi_i\}_{i=1}^r$ are the *POD modes* and $\{\alpha_i\}_{i=1}^m$ denote the *temporal coefficients* [56]. The temporal coefficients represent the time-dependent weights of the POD modes and describe how strongly each spatial mode contributes to the system state at time t.

A key advantage of POD is that its basis vectors are orthonormal [47]. Because of this, any error in the POD approximation splits into two independent parts: the projection error, which represents the part of the solution that the chosen modes fail to represent, and the prediction error, which comes from inaccuracies in the temporal coefficients. These two error components are orthogonal, meaning that the total squared error is simply the sum of the squared projection error and the squared prediction error. This orthogonality makes it possible to interpret and analyze the different sources of error separately. A proof of this decomposition, along with a discussion of how these error components arise in the ROM developed in this thesis, is provided in section 5.2

3.3.1. Relative information content

A key question is how many eigenvectors should be retained for an accurate approximation. In dynamical systems, large eigenvalues correspond to dominant system behavior, while smaller ones represent minor perturbations [56]. The relative information content of the reduced basis $\{\varphi_j\}_{j=1}^r$ is defined by

$$I(r) = \frac{\sum_{j=1}^{r} \lambda_j}{\sum_{j=1}^{m} \lambda_j}.$$

The number of modes r is chosen such that the basis captures nearly all variability in the snapshots, i.e.,

$$I(r) \approx 1.$$

For instance, one may select r such that 99% of the system is captured by the first r modes, which is equivalent to choosing the smallest r such that $I(r) > 0.99$ [6, 41].

3.3.2. Singular Value Decomposition (SVD)

The Singular Value Decomposition (SVD) is commonly employed to compute the POD modes. Given the snapshot matrix $X \in \mathbb{R}^{n\times m}$, its SVD is

$$X = U\Sigma V^T$$

where $U \in \mathbb{R}^{n\times n}$ and $V \in \mathbb{R}^{m\times m}$ are unitary matrices, and $\Sigma \in \mathbb{R}^{n\times m}$ is a diagonal matrix of the form

$$\Sigma = \mathsf{diag}(\sigma_1, \ldots, \sigma_l, 0, \ldots, 0)$$

with $l = \mathrm{rank}(X)$ and singular values ordered as $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_l > 0$.

Let the columns of U and V be denoted by $U = [u_1, ..., u_n]$ and $V = [v_1, ..., v_m]$. For each $i = 1, \ldots, r$, the following relations hold

$$\begin{aligned} Xv_i &= \sigma_i u_i \\ X^T u_i &= \sigma_i v_i \\ \sigma_i &= \sqrt{\lambda_i}, \end{aligned}$$

where λ_i is the i-th eigenvalue of the correlation matrix $XX^\top$. It then follows that

$$XX^T u_i = X\sigma_i v_i = \sigma_i^2 u_i = \lambda_i u_i, \qquad i = 1, \ldots, l.$$

This shows that the left singular vectors $u_1, \ldots, u_l$ are eigenvectors of $XX^\top$ and therefore correspond to the POD modes [56].

3.3.3. Computation of temporal coefficients

After establishing how the POD modes can be computed, the next question is how to determine the temporal coefficients $\{\alpha_i\}_{i=1}^m$. Assume we have a set of POD modes $\{\varphi_i\}_{i=1}^r$ and a snapshot matrix $X \in \mathbb{R}^{n\times m}$. As mentioned earlier, the POD approximation of the state at time t is given by

$$x_r(t) \approx \sum_{i=1}^{r} \alpha_i(t)\, \varphi_i.$$

Because the POD modes are orthonormal, the coefficients can be obtained as

$$\alpha_i(t) = \langle x_r(t), \varphi_i \rangle = \varphi_i^T x_r(t).$$

Galerkin projection However, when POD is used to predict the state of future time steps (i.e. in a ROM), these coefficients cannot rely on future snapshots. To address this, the Galerkin projection computes the temporal coefficients based on the POD modes and the governing equations [47].

Let V denote a Hilbert space, with $x_r(t) \in V_r \subset V$. The following stationary formulation is used solely to illustrate the general principle of Galerkin projection. Here $x(t) \in \mathbb{R}^n$ denotes the state vector. For a stationary problem of the form

$$a(x(t), v) = f(v) \quad \forall v \in V,$$

the bilinear form $a(\cdot, \cdot)$ represents the weak formulation of the underlying operator, while $f(v)$ denotes the corresponding forcing or input term evaluated against the test function v.

Projecting this equation onto the POD modes leads to the variational problem

$$a(x_r(t), v) = f(v) \quad \forall v \in V_r,$$

where V_r is the reduced space spanned by the POD modes. The Galerkin method seeks an approximate solution by enforcing the residual

$$R(v) := f(v) - a(x_r(t), v)$$

to be orthogonal to the reduced space V_r, i.e.

$$\langle R(v), w \rangle - 0 \quad \forall w \in V_r.$$

Since this space is spanned by the POD modes, we have

$$f(v) - a(x_r(t), v) = 0,$$

which simplifies to

$$a(x_r(t), v) = f(v) \quad \forall v \in V_r.$$

Substituting $x_r(t) = \sum_{i=1}^{r} \alpha_i(t)\, \varphi_i$ into the variational form gives

$$a\left(\sum_{j=1}^{r} \alpha_j \varphi_j, \varphi_i\right) = \sum_{j=1}^{r} \alpha_j\, a(\varphi_j, \varphi_i) = f(\varphi_i).$$

This results in a linear system

$$A\tilde{\alpha} = \tilde{b},$$

where $A \in \mathbb{R}^{r \times r}$ with entries $A_{ij} = a(\varphi_j, \varphi_i)$ and $\tilde{b} = [f(\varphi_1), \ldots, f(\varphi_r)]^T$. Solving this system for $\tilde{\alpha}$ yields the temporal coefficients [47].

Hybrid POD In addition to the intrusive Galerkin method, alternative approaches predict future values of temporal coefficients by using machine learning techniques. This can be achieved by forecasting temporal coefficients from their historical evolution, effectively propagating the time series [1, 8, 10, 22, 40, 50, 54, 66]. Another option is to train a neural network to predict the coefficients based on selected input features [26, 55, 59, 63, 65]. Since these methods do not rely on the governing equations, they are well-suited for complex or black-box systems. Applications of such hybrid strategies will be discussed in section 4.2.

3.4. Neural networks

This section aims to provide an introduction to the fundamental concepts of neural networks. It is based on the books *Deep Learning* by Goodfellow et al. (2016) [20] and *Neural Networks and Deep Learning* by C. Aggarwal (2023) [2].

Architecture

An artificial neural network is a machine learning technique which simulates the mechanism of learning in biological neurons in the human brain. These networks consist of nodes organized into layers: an input layer x, one or more hidden layers h_i, and an output layer y.

The *depth* of a network refers to the number of layers, while the *width* refers to the dimensionality of the hidden layers. Deep learning refers to neural networks with multiple hidden layers. Deeper

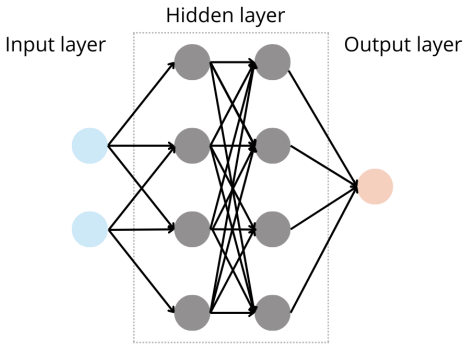


Figure 3.1: Visualization of a fully connected neural network

networks often require fewer units per layer and fewer parameters overall, however, they are also more challenging to optimize. When each node in a hidden layer is connected to all nodes in the subsequent layer, the layer is referred to as *fully connected*. An example of such an architecture is shown in figure 3.1.

In *feedforward* neural networks, information moves in one direction, from the input layer to the output layer, without any feedback loops where outputs are fed back into the network. When feedback connections are present, the architecture is known as a recurrent neural network (RNN). This work focuses solely on feedforward networks.

The goal of a neural network is to construct a function that relates inputs to outputs with the use of training examples. This is achieved by assigning weights to the connections between nodes, which represent the importance of each connection. During *training*, these weights are adjusted so that the network's predictions match the data. This iterative process of updating weights is referred to as *learning*. The term *hyperparameter* refers to parameters that govern the design of the model (e.g., the number of layers) or the training process (e.g., the learning rate), compared to the internal parameters of the model (e.g., weights) that are learned during training.

The neural network seeks to approximate a function f that maps the input x to the output y, i.e., $f(x) = y$. The network defines a mapping $y = \hat{f}(x;\theta)$ where θ represents the learnable parameters (weights and biases). The objective is to find θ such that $\hat{f} \approx f$.

To approximate both linear and nonlinear mappings, a neural network combines an affine transformation with a nonlinear *activation function*. Consider a deep neural network with n hidden layers. After transformation, the hidden layers can be expressed as

$$\begin{aligned} h_1 &= \alpha(W_1 x + b_1) \\ h_{i+1} &= \alpha(W_{i+1} h_i + b_{i+1}), \text{ for } i = 1, ..., n \end{aligned}$$

where x denotes the input layer, W and $b \in \mathbb{R}^n$ represent the weights and biases respectively, and α is the activation function. For regression tasks the output layer is typically fully linear. The *bias* term helps capture the invariant part of the prediction. Note that different activation functions can be used within the same network.

Cost function and gradient-based learning

Let the input and output datasets be defined as

$$\begin{aligned} \mathcal{X} &= \{x^{(1)}, ..., x^{(N)}\} \\ \mathcal{Y} &= \{y^{(1)}, ..., y^{(N)}\} \end{aligned}$$

where N denotes the number of data points. A neural network can be represented as a parameterized function

$$\hat{y}^{(i)} = \hat{f}(x^{(i)}; \boldsymbol{\theta}),$$

where $x^{(i)}$ is the i-th input and $\hat{y}^{(i)}$ is the corresponding predicted output. The goal of training is to find $\boldsymbol{\theta}$ such that $\hat{f}^{(i)}(x^{(i)};\boldsymbol{\theta}) \approx y^{(i)}$ for all $i = 1, \ldots, N$.

The *cost function* used by a machine learning algorithm is defined as

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \mathcal{L}(\hat{f}(x^{(i)};\boldsymbol{\theta}), y^{(i)}), \tag{3.1}$$

where $\mathcal{L}$ denotes a *loss function*. A common choice is the *Mean Squared Error (MSE)*, defined as

$$\mathcal{L}_{\mathsf{MSE}}(\hat{f}(x^{(i)};\boldsymbol{\theta}), y^{(i)}) = \frac{1}{N} \left\| \hat{f}(x^{(i)};\boldsymbol{\theta}) - y^{(i)} \right\|_2^2.$$

The choice of loss function depends on the predetermined task of the network, for example whether the network is used for classification or regression.

Training a neural network involves solving the optimization problem in equation (3.1). This is typically done using iterative, gradient-based optimizers that aim to minimize the cost function, in which in each iteration the network computes a prediction, evaluates the loss, and adjusts the parameters in a direction that reduces the loss. An *epoch* refers to processing the entire training dataset once, meaning that every training example has contributed to at least one parameter update that cycle. Gradients of the loss with respect to the network parameters are computed using *backpropagation*, an algorithm that applies the chain rule to propagate errors backward through the layers. The key idea is that the gradient indicates the direction of greatest increase in the loss, so updating the parameters in the opposite direction reduces the loss. The size of this update is controlled by the *learning rate* λ, which is another hyperparameter. A learning rate that is too large can cause the optimization process to diverge, while a value that is too small slows convergence [20].

In *batch gradient descent*, the gradient is computed using the entire training set, meaning the batch size equals N [2]. The training process can be broken down into the following steps:

Algorithm 2 Batch gradient descent

1: **Inputs:** dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, model $\hat{f}(\cdot;\boldsymbol{\theta})$, loss $\mathcal{L}$, learning rate λ, epochs E
2: **Initialize** parameters $\boldsymbol{\theta}$
3: **for** $e = 1$ to E **do** ▷ repeat over epochs
4: **Forward:** for all $i = 1, \ldots, N$, compute $\hat{y}^{(i)} \leftarrow \hat{f}(x^{(i)};\boldsymbol{\theta})$ ▷ full batch: uses all N samples
5: **Loss:** $\mathcal{L}_{\mathsf{full}} \leftarrow \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\big(\hat{f}(x^{(i)};\boldsymbol{\theta}), y^{(i)}\big)$
6: **Backpropagation:** compute gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathsf{full}}$
7: **Update:** $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda \cdot \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathsf{full}}$
8: **end for**
9: **Return** trained parameters $\boldsymbol{\theta}$

Activation functions

Activation functions introduce non-linearity into neural networks, enabling them to approximate nonlinear functions. The choice of activation function is not clear a priori and must be discovered through trial and error. Early neural networks commonly employed classical activation functions such as the *sign*, *sigmoid* and *hyperbolic tangent (tanh)* functions. The sign function is defined as

$$\alpha(z) = \mathsf{sign}(z) = \begin{cases} -1 & \text{if } z < 0, \\ 0 & \text{if } z = 0, \\ 1 & \text{if } z > 0, \end{cases}$$

and is useful for binary outputs during prediction. However, its non-differentiability makes it unsuitable for gradient-based optimization methods. The sigmoid activation function is given by

$$\alpha(z) = \frac{1}{1 + e^{-z}},$$

while the tanh function is defined as

$$\alpha(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}.$$

These two functions satisfy

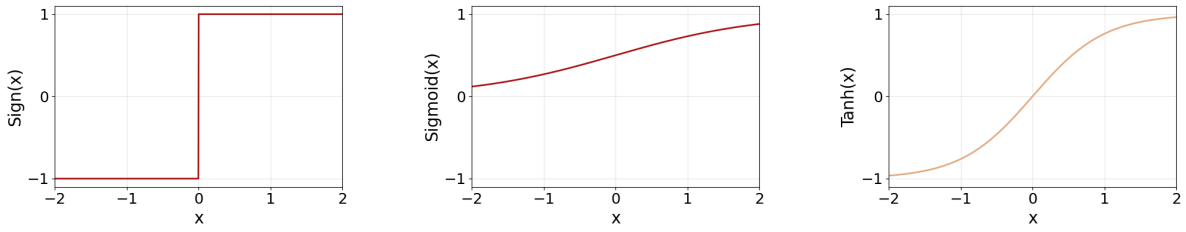
$$\tanh(z) = 2\sigma(2z) - 1,$$

showing that tanh is essentially a rescaled and shifted version of the sigmoid.

Furthermore, for large $|z|$

$$\tanh(z) \approx \mathsf{sign}(z)$$

meaning that tanh behaves similarly to the sign function but remains differentiable, providing non-zero gradients for most inputs and is thus more suitable for gradient-based training compared to sign. Both sigmoid and tanh, however, suffer from saturation for large input values, where their derivatives approach zero. This can hinder learning in deep networks due to vanishing gradients. For this reason, it is generally discouraged to use in hidden layers, though it remains compatible as an output unit when paired with an appropriate loss function. The tanh function is often favored over the sigmoid when both positive and negative outputs are desired [2].

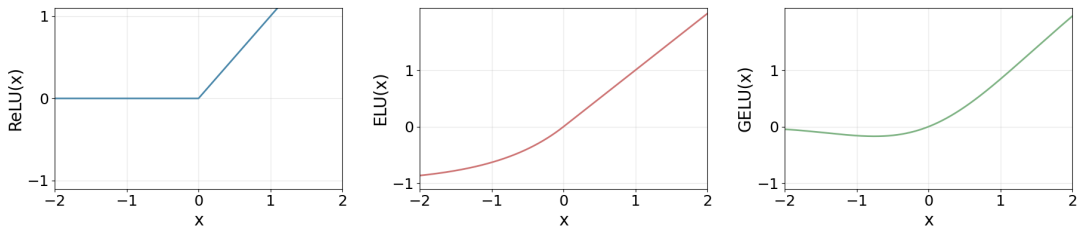


(a) Sign **(b)** Sigmoid **(c)** Tanh

Figure 3.2: Plots of activation functions

To avoid vanishing gradients, piecewise linear activation functions are often used. The *Rectified Linear Unit (ReLU)* is one of the most widely used and is defined as

$$\alpha(z) = \max\{0, z\}. \tag{3.2}$$



(a) ReLU **(b)** ELU **(c)** GELU

Figure 3.3: Plots of activation functions

ReLU offers a useful gradient direction for learning, as its derivative is 1 when the unit is active and 0 otherwise. A drawback, however, is that units receiving only non-positive inputs produce zero activations and therefore stop updating. In certain architectures such as recurrent networks, probabilistic models, and autoencoders, the use of sigmoidal units may still be preferred due to specific structural requirements which rule out the use of piecewise linear activation functions [2].

Because ReLU is non-differentiable at $z = 0$ and outputs only non-negative values, several smooth alternatives have been proposed. The *Exponential Linear Unit (ELU)* replaces the negative half of the

activation with a smooth exponential curve, allowing small negative outputs and providing non-zero gradients for negative inputs [9]. ELU is defined as

$$\alpha(z) = \begin{cases} \alpha_i(e^x - 1) & \text{if } z \leq 0, \\ z & \text{if } z > 0, \end{cases} \tag{3.3}$$

where α_i is a hyperparameter that controls the saturation level for negative inputs and is commonly set to 1. The *Gaussian Error Linear Unit (GELU)* provides a smoother alternative to ReLU by weighting inputs according to their probability under a Gaussian distribution [25]. This allows the activation to transition gradually rather than abruptly at zero. GELU is defined as

$$\alpha(z) = z\Phi(z), \tag{3.4}$$

where $\Phi(z)$ is the standard normal cumulative distribution function.

3.5. Performance metrics

In this section, the main criteria for evaluating the performance of the ROM developed in this work will be presented, including validation strategies, structure preservation, accuracy metrics, and computational speed-up.

3.5.1. Validation

When evaluating the performance of a ROM, it is important to test the model on data that was not used during training. A common way to achieve this is by dividing the available dataset into three subsets: training, validation, and test sets. The training set is used to learn the model weights through optimization techniques. However, hyperparameters, which govern the overall structure and behavior of the model, should not be tuned using the same data employed for training. Instead, a separate portion of the data is reserved as a validation set, which allows for evaluating the model's performance under different hyperparameter configurations. Once the model has been trained and its hyperparameters optimized, the test set is used to assess its final performance. This evaluation provides an estimate of the model's generalization ability. The relative sizes of the training, validation, and test sets depend on the total amount of available data and are typically determined by the researcher based on the specific requirements of the task [2].

Cross-validation consists of repeatedly partitioning the data into multiple folds, training the model on a particular set of folds and validating it on the remaining ones, so that each fold serves as a validation set at least once. The results are then averaged across all folds. This technique is particularly useful for hyperparameter tuning as it reduces the risk of overfitting to a specific split and provides a clearer picture of how the ROM will behave on unseen data [2].

Early stopping is a regularization technique that stops the neural network training when the model stops improving on the validation set, thereby preventing overfitting. The patience parameter specifies how many additional epochs the training is allowed to continue without improvement before stopping. A larger patience lets the optimizer explore temporary plateaus, while a smaller patience stops training more aggressively.

3.5.2. Structure preserving reduction

In addition to the metrics used in this work, the ROM literature frequently considers structure-preserving properties. These concepts are summarized here because they are referenced in the related work discussion, although they are not evaluated in the present study.

- *Passivity* ensures that the system cannot generate energy and can only dissipate the energy initially stored. Passivity is crucial in electrical networks and control systems, as non-passive models may lead to instability when interconnected. Certain techniques, such as structure-preserving projection methods, are known to enforce passivity during MOR [39].
- *Stability* refers to the system's response to small perturbations in initial conditions. A system is considered stable if small changes in the input do not cause large deviations in the output

over time. In continuous-time dynamical systems, stability is often analyzed using the concept of *Lyapunov stability*. For an autonomous system $\dot{x} = f(x)$ with equilibrium at x_e (i.e., $f(x_e) = 0$), the system is Lyapunov stable if, for every $\epsilon > 0$, there exists a $\delta > 0$ such that $\|x(0) - x_e\| < \delta$ implies $\|x(t) - x_e\| < \epsilon$ for all $t \geq 0$. If, in addition, $\lim_{t\to\infty} \|x(t) - x_e\| = 0$, the system is said to be *asymptotically stable*. A common approach to verify this is through a *Lyapunov function* $V : \mathbb{R}^n \to \mathbb{R}$, which satisfies

1. $V(x) = 0$ if and only if $x = x_e$,
2. $V(x) > 0$ for all $x \neq x_e$,
3. $\dot{V}(x) = \nabla V \cdot f(x) \leq 0$ for all x.

If in addition $\dot{V}(x) < 0$ for all $x \neq x_e$, the system is asymptotically stable [3].

- A *Port-Hamiltonian* system is defined by its energy-based formulation, where energy flows through designated ports that interact with other subsystems. This structure guarantees passivity and often also implies stability. One of the key advantages of the port-Hamiltonian framework is its closure under power-preserving interconnections, making it particularly suitable for modeling complex networks and control purposes. Consequently, preserving the port-Hamiltonian structure in a ROM not only maintains physical consistency but also helps ensure passivity and, in many cases, stability of the reduced system [57].

3.5.3. Accuracy metrics

The following section presents a set of metrics designed to evaluate the accuracy of the ROM in comparison to the FOM.

Error metrics

A fundamental requirement for any ROM is to approximate the FOM as accurately as possible. To quantify the discrepancy between the two, various error metrics can be employed depending on the context and the properties of interest [29].

- *Mean Squared Error (MSE)*: This metric is commonly used when comparing time-series data or outputs over a discrete set of time steps. It is defined as

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} \|x_{\text{FOM}}^{(i)} - x_{\text{ROM}}^{(i)}\|_2^2, \tag{3.5}$$

  where $\|\cdot\|_2$ denotes the L_2 norm and N is the number of data points. MSE is useful for assessing average performance over time.

- *Frobenius norm*: The Frobenius norm measures the scalar difference between two matrices. It is defined as

$$\text{Frobenius norm} = \|X_{\text{FOM}} - X_{\text{ROM}}\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} \left(X_{\text{FOM}}^{(i,j)} - X_{\text{ROM}}^{(i,j)}\right)^2}, \tag{3.6}$$

  where $X_{\text{FOM}}, X_{\text{ROM}} \in \mathbb{R}^{n\times m}$. In addition, the *relative* Frobenius norm is defined as

$$\text{relative Frobenius norm} = \frac{\|X_{\text{FOM}} - X_{\text{ROM}}\|_F}{\|X_{\text{FOM}}\|_F}. \tag{3.7}$$

- *Mean relative error*: This norm-based metric is defined as

$$\text{mean relative error} = \frac{1}{N} \sum_{i=1}^{N} \frac{\|x_{\text{FOM}}^{(i)} - x_{\text{ROM}}^{(i)}\|_2}{\|x_{\text{FOM}}^{(i)}\|_2}, \tag{3.8}$$

  where $\|\cdot\|_2$ is the L_2 norm and N is the number of data points. This metric is scale-invariant and useful when comparing solutions across different magnitudes.

It is expected that some accuracy will be lost during reduction; however, the acceptable level of deviation between the ROM and the FOM typically depends on the specific application and its performance requirements. Researchers therefore define acceptable error bounds that reflect these constraints. The specific error bound considered acceptable for this work will be discussed in section 5.2.

3.5.4. Computational speed

As ROMs are primarily designed to accelerate simulations compared to FOMs, evaluating their computational efficiency and execution time is essential.

Speed-up factor

A commonly used metric to quantify computational gain is the *speed-up factor*, defined as

$$\text{speed-up} = \frac{T_{\text{FOM}}}{T_{\text{ROM}}}, \tag{3.9}$$

where T denotes the execution time of the respective models [1]. A higher speed-up factor indicates a more efficient ROM in terms of computational time. It is important to note that this value can vary depending on hardware specifications (see chapter 6 for details relevant to this work).

The execution time of a ROM can be divided into two phases: the *offline* phase and the *online* phase [45]. The offline phase consists of the construction of the ROM, which may involve generating snapshots from the FOM, applying dimensionality reduction techniques or training a neural network. Although this stage can be computationally expensive, it is performed only once. In contrast, the online phase uses the precomputed ROM to perform simulations or predictions for new inputs, parameters, or time steps. This reusability makes ROMs particularly beneficial for real-time applications where fast adaptation to new scenario is required, like in optimization [44]. In this work, we solely consider the speed-up factor of the online phase; the offline training times will be reported separately.

4

Related work

The aim of this chapter is to review existing research relevant to the development of ROMs for DH networks. Section 4.1 examines prior work in which ROMs were designed and implemented specifically for DH systems, distinguishing between intrusive and non-intrusive approaches. Section 4.2 focuses on studies applying the hybrid POD method, which combines POD-based modal decomposition with techniques for predicting the corresponding temporal coefficients. Within this context, two strategies for coefficient prediction are discussed. Additional technical details of the ROM methods referenced in this chapter are provided in appendix A.

4.1. Review of ROM approaches for DH networks

Extensive research has focused on modeling DH networks [12, 15, 18, 35, 38, 51], on reducing network complexity through thermal load clustering [31, 36, 37, 58], and on forecasting thermal demand profiles [11, 15, 23, 24]. In contrast, comparatively few studies have investigated the use of ROM techniques to accelerate numerical simulations or optimal control computations of DH networks [5, 19, 22, 28, 51, 53], which will be discussed in this chapter.

Intrusive method For his PhD thesis [51], Markus Rein models the DH network dynamics using hyperbolic PDEs to describe the mass and energy transport and algebraic constraints to satisfy the conservation of mass law and pressure balances. He combines several intrusive ROM techniques to develop a surrogate model for the efficient simulation of DH networks. First, the volume flow is fixed at certain values. This step allows the full nonlinear PDE system to be locally linearized. For each local linearization, a transfer function (A.1) is computed. This function describes how system inputs (energy density and flow at the source) affect the desired output (energy density at consumer nodes). To approximate this input-output behavior, Rein uses moment matching (for more details see A.2). Moments are derivatives of the transfer function evaluated at selected frequencies. The matching of these moments ensures that the ROM reproduces the behavior of the original system in the frequency domain. A greedy algorithm then determines which local linearizations contribute most to the global error reduction. The union of these reduced spaces results in a global space. This is further refined using SVD to remove redundant basis vectors. Finally, Rein applies a one-sided Petrov-Galerkin projection (for details see A.3) where the test space is the reduced basis space with an energy-weighted inner product, which is known to preserve passivity and Lyapunov stability in port-Hamiltonian systems [42].

The ROM was tested on two cases, a street (32 consumers, 81 pipe segments) and a city (333 consumers, 775 pipe segments), and even though he does not specify the dimension of the ROM, he achieves up to $10\times$ speed-up of computation time compared to the FOM. Rein also extends it to an optimal control problem, achieving a minimum $4\times$ speed-up in the optimization phase for the city case. The maximum relative L_2 error was $5,28 \times 10^{-3}$ when compared to the FOM.

However, Rein introduces several simplifying assumptions to make his approach feasible. First, heat losses are completely neglected, which removes the source term from the internal energy balance

equation and ensures strict energy conservation. This is essential for embedding the system into a port-Hamiltonian framework, as such systems rely on energy conservation to guarantee passivity and Lyapunov stability. While this assumption simplifies stability analysis, it deviates significantly from real-world conditions, making it problematic for Gradyent's case where heat losses are critical. Second, all pipeline segments are assumed to have identical diameter and length. This uniformity eliminates geometric variability in the governing PDE coefficients, simplifying local linearization and making the computation and interpolation of transfer functions during moment matching more straightforward. Finally, the network is restricted to a single source, which greatly reduces the complexity of the algebraic constraints imposed at nodes, namely the mass conservation and pressure balance, but again differs greatly from realistic DH systems.

Non-intrusive methods As described in section 3.2, in some cases the governing equations of a FOM are unavailable, such as in black-box models or systems that are too complex to be accurately described by a closed set of equations. This means that a non-intrusive approach is required to construct a ROM. Another reason why this approach is preferred is when intrusive methods, such as computing temporal coefficients via the Galerkin projection (see section 3.3.3) of the FOM, are too computationally expensive. Since the cost of these projections scales with the FOM dimension, they may not offer any real computational speed-up [26].

A common approach in ROM methods is to begin with analyzing heat transfer through water flow in a single pipe segment, represented as an edge in the broader network. Jiang et al. [28] derive a FOM which incorporates the radii and material properties of the various pipe components. They then construct a non-intrusive ROM that predicts the outlet water temperature based on two inputs: the inlet water temperature and the ground temperature. The ROM was tested in two scenarios: one with a single producer and two users, and another with a closed-loop control system. While the ROM achieved high accuracy in predicting outlet temperatures, its computational speed-up was limited when using the same time steps as the FOM, only $\sim 1.2\times$ faster. Substantial gains, however, are expected for larger and more complex networks, when speed-up can reach orders of magnitude. This expectation is debated by [15], who argues that the computational time for simulating a small network with few consumers cannot be directly extrapolated to larger systems because the relationship between computational cost and network size is highly nonlinear.

Guelpa et al. [22] focus exclusively on optimizing pumping energy in large-scale DH networks. To address this, they construct a ROM based on POD. The temporal coefficients associated with the POD modes are then obtained through interpolation from precomputed training simulations. This ROM is embedded into an optimization framework that minimizes pumping energy under realistic operational constraints. Even though they do not report the size of the snapshot matrix or the number of retained modes, the results demonstrate that the POD-based ROM predicts optimal pumping costs as a function of thermal load with average relative errors of approximately 5%. Furthermore, the approach achieves more than an 80% reduction in simulation time and reduces pumping costs by around 20%.

In our work, POD can likewise be used to obtain a compact representation of the snapshot matrix containing all relevant state variables. However, the coefficient prediction strategy employed by Guelpa et al. is not applicable. This is because Gradyent's optimization is formulated as a time-independent problem, meaning there is no temporal trajectory to interpolate.

Using neural networks to accelerate DH network simulations has been explored before, often incorporating physical topology. Boussaid et al. [5] employ a graph neural network (GNN), a class of models designed to operate directly on graph-structured data by propagating information along edges and aggregating features from neighboring nodes. Their GNN simultaneously learns a representation of the network's structure and performs node-level regression, enabling it to predict forward and return pressures and temperatures at all nodes. Initially tested on a small network with a single source, the approach was later scaled to a network with 50 thermal loads to assess scalability. The proposed GNN achieves high accuracy, with average RMSEs (A.2) of $\sim$0.6% for temperature, <0.1% for forward pressure, and 2% for return pressure. Computational time is reduced by 99% compared to the FOM, and this advantage grows with network size, while accuracy remains robust (e.g., roughly 2.3% RMSE for temperature in network with 100 nodes).

Similarly, Dubon et al. [53] propose a topology-reduction framework that replaces entire substation clusters with surrogate neural networks. This produces a simplified network graph in which the substituted clusters function as single aggregated nodes. Each surrogate model predicts the outgoing flow temperatures of the nodes within its cluster based on incoming temperatures and individual heat demands. To ensure diversity, 220 clusters were selected for training, varying in size, topology, and demand profiles. The method achieved a 27% reduction in overall simulation time by replacing approximately 39% of consumer nodes, while maintaining high accuracy with RMSEs in the range of 0.1-0.6%.

De Giuli et al. [19] explore neural network based optimal control for DH systems by introducing a physics-informed neural network (PINN). Traditional PINNs embed physical constraints into training by minimizing not only the discrepancy between predictions and observations, but also the residuals of the governing PDEs [49]. However, de Giuli et al. do not apply PINNs in this standard way; instead, they embed the physical topology of the DH network directly into the neural network architecture. Each network segment, pipe in the forward network and a thermal load, is represented by an individual neural network, and these are interconnected according to the actual network graph. The control objective is formulated as minimizing the production cost at the heating station. In their test case (network with one source and five demand nodes) the PI-RNN achieved a fit of 81%, compared to 51% for a standard neural network, and reduced training time from 145 minutes to 91 minutes. However, prediction time is not reported, and no explicit error metrics for the predictions are provided.

The neural network models described above are inherently non-generalizable: a trained model cannot be directly applied to a different DH network. More importantly, adapting these approaches to a new network is not a uniform or automated process. The number and structure of neural networks required depend on the size and complexity of the network. Consequently, each implementation demands a case-specific investigation to determine the appropriate architecture and training strategy, significantly increasing effort and scalability challenges.

4.2. Hybrid POD

Hybrid POD follows the non-intrusive framework by first applying POD to perform a modal decomposition and identify the dominant spatial modes. The temporal evolution of the corresponding modal coefficients is then predicted using intrusive, data-driven methods. Once the future coefficients are predicted, the system state at future times can be reconstructed using the previously computed POD modes. Because this approach does not depend on the governing equations, it is well suited for complex or black-box systems. Moreover, by incorporating data-driven models for the temporal dynamics, hybrid POD can capture nonlinear behavior more effectively than standard POD-based ROMs [50].

Hybrid POD has been successfully applied across a range of applications, particularly in fluid and flow-related problems, such as cavitating flows [26, 65, 66], reacting flows [10, 63], aerodynamic flows [40, 59], fluid in a circular cylinder [1, 54], blood flow [50], unsteady fluid flows [8] and internal flow within a rocket motor [65].

Time-series propagation of POD coefficients An important distinction within hybrid POD concerns the method used to compute or propagate the temporal coefficients. A commonly adopted approach is to forecast future temporal coefficients based on their historical values, effectively treating the coefficients as a time series [1, 8, 10, 40, 50, 54, 66], as also seen in Guelpa et al.'s approach for optimizing pumping energy in DH networks [22]. This approach is particularly suitable for time-dependent simulations.

Despite the computational speed-up for these time-propagation-based hybrid POD models in the literature mentioned above, this approach is not suitable for the construction of the ROM in this work. As mentioned earlier, the intended application of the ROM is within an optimization framework that depends solely on the current state variables of the network, without reference to any previous states, and is therefore inherently time-independent. Consequently, forecasting the temporal evolution of the POD coefficients is not meaningful in this context. Instead, the coefficients must be predicted directly as functions of the input conditions provided to the network state estimator solver.

Predicting POD coefficients based on input conditions This other non-intrusive option consists of using machine learning to learn the mapping from the input conditions of the FOM to the POD temporal coefficients [26, 55, 59, 63, 65]. For this purpose, feedforward neural networks are most commonly used [26, 63, 55, 59]. Xu et al. [65] compared several architectures and found that while all achieved significant computational speed-up, a self-attention-based neural network delivered the highest accuracy relative to the FOM. However, a feedforward network ranked second on both accuracy and speed, performing close to the self-attention model.

Salvador et al. [55] proposed to use kernel POD instead of traditional POD. Kernel methods map the original data into a higher-dimensional feature space using nonlinear transformations, which means the POD can capture more complex, nonlinear relationships (for details see A.4). This results in a more accurate approximation of the snapshot set while maintaining similar efficiency to standard POD, particularly beneficial for highly nonlinear cases such as reacting flows. Additionally, they dynamically adapted the network architecture, varying the number of hidden layers and neurons based on the size of the reduced space, which improved accuracy for more complex reduced bases.

Another motivation for learning the mapping from input conditions rather than propagating the time series, is that standard interpolation techniques often fail when only a limited number of samples are available. Wang et al. [63] faced a time-dependent problem with insufficient data for propagation. To address this, they included the time step as an additional input to the neural network, alongside the parameter inputs, to capture temporal dynamics.

Overall, the existing literature shows that, within the test cases considered, predicting POD temporal coefficients from parameter inputs using neural networks is both accurate and efficient, consistently achieving computational speed-up. The extent of this speed-up and the design choices for the neural network depend on the specific test case. In general, feedforward networks are the simplest to implement and serve as a practical starting point. They can be extended or refined when needed, for example by incorporating time dependency or adapting network size to the complexity of the reduced space.

The limitations of the studies reviewed in this chapter show that existing approaches reveal a clear research gap: there is currently no ROM framework for DH networks that avoids physical simplifications, does not require topology-specific neural network architectures, and can be retrained automatically for different networks using FOM data. Addressing this gap directly relates to the first research subquestion of this thesis, which asks how ROM methods and neural networks can be combined to construct a retrainable ROM without manual adaptation. The next chapter develops a methodology that fills this gap by integrating POD-based dimensionality reduction with a topology-independent neural network mapping from input conditions to POD coefficients.

5

Methodology

This chapter provides an overview of the methodology used to develop the ROM. It begins with section 5.1, which explains the reasoning behind the design choices. Section 5.2 then offers a detailed description of the ROM framework, highlighting the differences between the training and prediction phase and discussing its evaluation. The first subquestion, *"How can ROM methods be combined with neural networks to construct a ROM that can be retrained for different DH networks without manual adaptation of the model architecture?"*, is addressed in this section.

5.1. Design motivation

As discussed in the previous chapter, Rein adopted an intrusive approach to apply MOR to his DH model [51]. However, this method relies on restrictive assumptions, namely neglecting heat losses, assuming uniform pipe dimensions, and considering single-source networks. Because these assumptions oversimplify the physical realities of DH networks; they cannot be applied to the complex systems that Gradyent works with.

Existing non-intrusive approaches often employ neural networks that incorporate the topology of the network [5, 19, 53]. While promising, these methods face significant limitations: each DH network requires manual architectural design before training. The number of neural networks, their connectivity, and their internal structure depend directly on the size and complexity of the DH network. As a result, every new implementation demands human analysis to determine an appropriate architecture and training strategy, which prevents automated retraining.

These limitations show that existing MOR techniques are not directly applicable to multiple realistic DH networks because their training pipelines cannot be reused without topology-specific redesign. This highlights a clear research gap: the need for MOR strategies that can compute key state variables while remaining adaptable to the complexity of real-world DH systems and that rely on a fully automated training process based solely on FOM data rather than topology-specific intervention for each new network configuration.

As outlined in section 2.3, the network state estimator is computationally intensive due to its iterative state update process. There are three potential strategies to accelerate this step:

1. reducing the number of iterations,
2. reducing the time per iteration,
3. replacing the iterative process entirely.

Replacing the entire iterative process is challenging because the model's key variables are nonlinearly interdependent, which is precisely why iterative solving was required in the first place. Moreover, every solution must remain physically consistent: the FOM cannot, for example, produce a state where a node outputs more flow than it receives. In principle, one could design a ROM that enforces these physical constraints directly [51]. However, this would require the ROM architecture to be closely tied

to the structure of the DH network, which would make the training process system dependent on the DH network architecture.

A speed-up opportunity lies in providing a better initial estimate for the iterative process. The FOM normally performs a preliminary iteration to construct this initial estimate. When the ROM is used to generate the initial guess, instead, this preliminary iteration is no longer needed. Furthermore, if the ROM prediction is already closer to the optimal solution than the initial guess produced by the full model, the subsequent state iteration procedure may require fewer iterations to converge. In practice, convergence typically requires between 2 and 10 iterations, depending on the DH network and the evaluated optimization candidate.

The resulting speed-up therefore stems both from removing the initial guess iteration and from the possibility that the ROM prediction is closer to the optimal, reducing the total number of iterations needed.

The ROM is therefore tasked with predicting the optimal state that the iterative solver would normally converge to. Although the governing physical equations are known, the coupled nonlinear interactions between variables mean that the state cannot be obtained without an iterative solution procedure. This iterative process cannot be rewritten in a form that allows intrusive projection or reduction. Therefore, a non-intrusive approach is the only viable option for applying MOR to the network state estimator.

The proposed methodology adopts a hybrid POD approach, where POD compresses the state data into spatial modes, and a neural network predicts the temporal coefficients. The objective of the developed ROM is to accelerate the optimization process. This process is entirely time-independent, meaning it does not rely on previous states but instead considers only the current state variables of the DH network. Consequently, propagating the temporal evolution of the POD coefficients is not feasible; they must be computed directly from the given input conditions. To achieve this, the neural network is designed to learn a mapping from these input conditions to the corresponding temporal coefficients.

An important limitation of the ROM is that its predictions are not guaranteed to be physically consistent. For this reason, the ROM output is not used as a final state, but rather as an initial guess within the existing state iteration procedure. The ROM provides the initial guess, and the subsequent iterations ensure convergence to a physically consistent solution.

Neural networks are chosen over other machine learning techniques for their ability to effectively capture strong non-linearities. The inclusion of POD introduces an intermediate error, offering more interpretability and control over the modeling process. In contrast, pure neural network models function as black boxes, making it harder to diagnose performance issues. Moreover, reducing dimensionality prior to training is expected to accelerate the learning process and decrease data requirements, potentially making the approach more efficient than standard neural network based methods.

A feedforward neural network architecture has been chosen for this approach. This design is straightforward and provides an effective starting point for initial testing. Moreover, since the optimization process treats each time step independently, there is no need for the network to learn recurrent connections as the state of the previous time step does not influence the current one. More details about the design and hyperparameters of the neural network for the two test cases can be found in tables 6.5 and 7.5.

For each new DH network, POD must be recomputed (as it depends on network-specific snapshot data), and the neural network retrained. However, this retraining can follow the exact same standardized pipeline, with the architecture remaining fixed and only the FOM data changing. This stands in clear contrast to existing topology-dependent neural network approaches, which require human analysis of each network's structure to design a suitable architecture. In those methods, the number of clusters, their connectivity, and the overall model structure must be adapted manually for every new network, making the training process non-uniform [5, 19, 53].

5.2. ROM architecture and framework

The hybrid POD method consists of two distinct phases: an offline phase and an online phase. During the offline phase, data is collected, dimensionality reduction is performed, and the neural network is trained. In the online phase, the trained model is employed to predict a new system state based on

a given input condition. A schematic overview of the training and prediction process can be found in figure 5.1, which illustrates how POD is integrated with a neural network to construct a ROM with a general, topology-independent training pipeline based on FOM data.

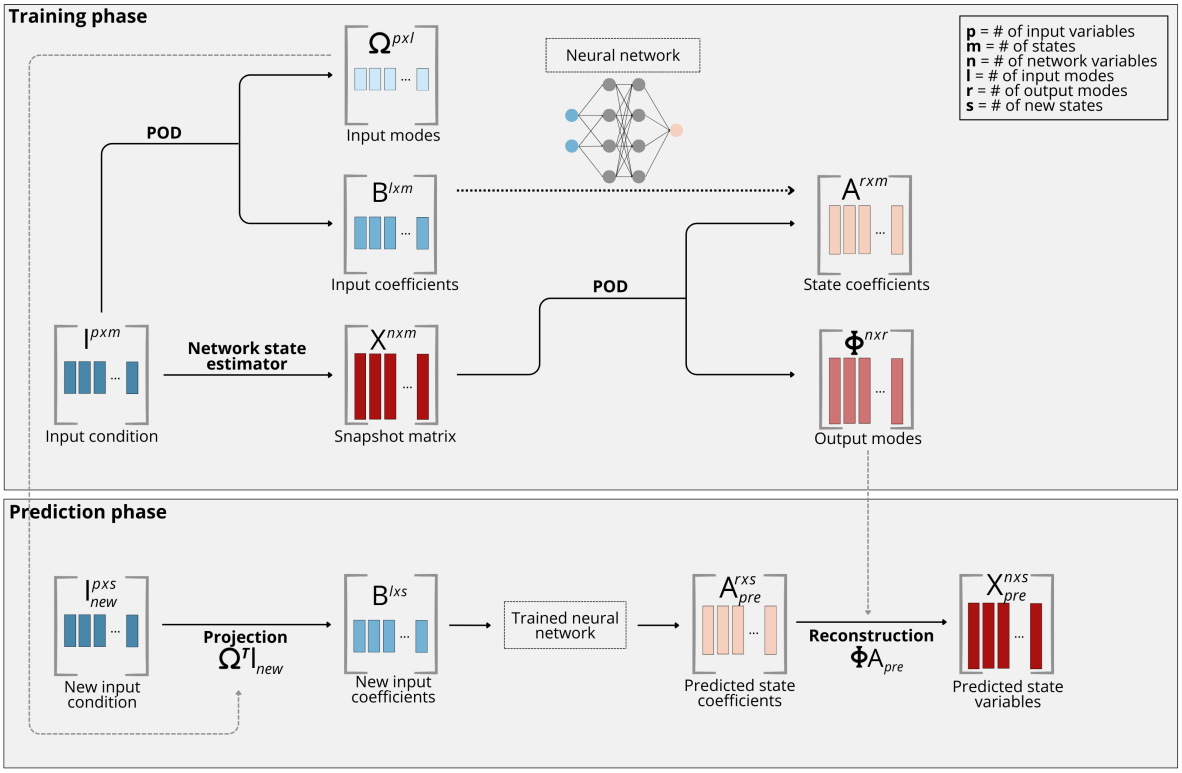


Figure 5.1: Framework of hybrid POD architecture

Training Training begins with the network state estimator, which, given a specific set of input conditions, computes the state variables for every component in the DH network at each instance (see section 2.3 for details). These results are organized into a snapshot matrix, where each column represents a single instance and each row corresponds to the value of a variable at a specific component. Applying POD to this snapshot matrix results in a set of output POD modes. The temporal coefficients are obtained by projecting the data onto the retained POD modes. Mathematically, this projection is expressed as

$$A = \Phi^T X,$$

where X denotes the corresponding snapshot matrix for each set, Φ represent the POD modes and A denotes the temporal coefficients. The neural network is then trained to learn a mapping from the input conditions to these temporal coefficients.

However, the input conditions themselves can be highly dimensional. To address this, POD is applied a second time to the input conditions, yielding input modes and input coefficients. These input coefficients serve as the neural network's inputs, allowing it to learn per instance while working with a more compact representation. Both the input and output modes are computed once during training and remain fixed for all subsequent predictions.

Before training, the input coefficients are normalized to zero mean and unit variance; the predicted coefficients are denormalized afterwards. The neural network is trained using the MSE loss (3.5) between the predicted and true temporal coefficients. All remaining hyperparameters and training settings used for the two test networks are summarized in table 6.5 and 7.5.

The training data originates from a process known as historical optimization runs performed when a new project begins at Gradyent. In these runs, previously observed boundary conditions are used as inputs to the network state estimator, which computes the corresponding state variables. Because

this procedure is already part of the onboarding pipeline for new clients, a sufficient amount of data is readily available for training. More details on the size of the datasets used in this work can be found in sections 6.2 and 7.1.

Prediction The online phase involves generating predictions for instances that the neural network has not encountered during training, based on a new input condition. To ensure compatibility with the neural network's expected input format, the new input condition is projected onto the precomputed input POD basis to obtain the corresponding input coefficients. These input coefficients are then fed into the trained neural network, which predicts the associated state coefficients. Using these predicted coefficients together with the precomputed output modes, the full set of predicted state variables can be reconstructed.

Evaluation We distinguish four types of errors that arise throughout the reduction and prediction pipeline. Let X denote the snapshot matrix, and let its POD decomposition be given by the output modes Φ and temporal coefficients A. Similarly, let I be the input condition matrix, whose POD decomposition consists of the input modes Ω and the corresponding input coefficients B. Based on these quantities, we define the

- input projection error: $e_{\text{in,proj}} = I - \Omega B$,
- output projection error: $e_{\text{out,proj}} = X - \Phi A$,
- prediction error: $e_{\text{pred}} = \Phi A - \Phi A_{\text{pred}}$,
- total error $e_{\text{total}} = X - \Phi A_{\text{pred}}$.

Here, A_{pred} denotes the temporal coefficients predicted by the neural network.

All errors except the input projection error are expressed in the space spanned by the output POD modes. This ensures that the output projection, neural network, and total error are directly comparable.

Furthermore, since we have that

$$\begin{aligned} e_{\text{total}} &= X - \Phi A_{\text{pred}} \\ &= (X - \Phi A) + (\Phi A - \Phi A_{\text{pred}}) \\ &= e_{\text{out,proj}} + e_{\text{pred}}, \end{aligned}$$

the total reconstruction error decomposes into the output projection error and the prediction error. The POD modes form an orthonormal basis of the reduced subspace. Consequently, the projection error lies in the orthogonal complement of this subspace, while the prediction error lies entirely inside it.

By the defining property of the POD projection,

$$\Phi e_{\text{out,proj}} = 0.$$

We obtain

$$\begin{aligned} \langle e_{\text{out,proj}}, e_{\text{pred}} \rangle_F &= \langle e_{\text{out,proj}}, \Phi A - \Phi A_{\text{pred}} \rangle_F \\ &= \langle e_{\text{out,proj}}, \Phi(A - A_{\text{pred}}) \rangle_F \\ &= \langle \Phi^T e_{\text{out,proj}}, A - A_{\text{pred}} \rangle_F \\ &= \langle 0, A - A_{\text{pred}} \rangle_F = 0, \end{aligned}$$

which proves the orthogonality of the two error components.

Since the errors lie in orthogonal subspaces, their squared Frobenius norms satisfy:

$$\|e_{\text{total}}\|_F^2 = \|e_{\text{out,proj}} + e_{\text{pred}}\|_F^2 = \|e_{\text{out,proj}}\|_F^2 + \|e_{\text{pred}}\|_F^2 + 2\|e_{\text{pred}}\|_F \|e_{\text{out,proj}}\|_F = \|e_{\text{out,proj}}\|_F^2 + \|e_{\text{pred}}\|_F^2.$$

Both the absolute and relative Frobenius norms of each error will be reported. The absolute Frobenius norm (3.6) quantifies the true magnitude of the reconstruction error and reveals how projection and prediction errors contribute to the total error through the orthogonal decomposition above. Because the

power variables in the dataset have very large magnitudes (typically $10^3 - 10^5$), the absolute Frobenius error can appear large even when the relative reconstruction quality is good. This makes it difficult to evaluate model accuracy solely based on absolute values. For this reason, we additionally report the relative Frobenius error (3.7), which normalizes the reconstruction error by the overall magnitude of the data and therefore provides a global measure of accuracy.

In addition to the Frobenius-based metrics, we also report the mean relative L_2 error defined in (3.8). While the relative Frobenius norm gives a global assessment of reconstruction quality, the mean relative L_2 error evaluates reconstruction accuracy per state variable over the different instances, therefore giving more insights into localized inaccuracies.

The output POD basis is constructed to optimally approximate entire snapshots, i.e. full system states at a given instance. Therefore, the output projection error is evaluated per snapshot and the mean is taken over the snapshots. The input projection error is evaluated similarly.

The prediction and total error are computed row-wise because these metrics assess how accurately each individual state variable is reconstructed over time, which reflects the quality of the predicted temporal coefficients. This results in the following explicit formulations:

$$\text{input projection error} = \frac{1}{m}\sum_{i=1}^{m}\frac{\left\|I^{(i)} - \Omega B^{(i)}\right\|_2}{\left\|I^{(i)}\right\|_2}, \tag{5.1}$$

$$\text{output projection error} = \frac{1}{m}\sum_{i=1}^{m}\frac{\left\|X^{(i)} - \Phi A^{(i)}\right\|_2}{\left\|X^{(i)}\right\|_2}, \tag{5.2}$$

$$\text{prediction error} = \frac{1}{n}\sum_{j=1}^{n}\frac{\left\|e_j^T(\Phi A - \Phi A_{\text{pre}})\right\|_2}{\left\|e_j^T X\right\|_2}, \tag{5.3}$$

$$\text{total error} = \frac{1}{n}\sum_{j=1}^{n}\frac{\left\|e_j^T(X - \Phi A_{\text{pre}})\right\|_2}{\left\|e_j^T X\right\|_2}. \tag{5.4}$$

The objective of this study is to achieve a mean relative error below 5% for each of the four error types. Modeling uncertainties within this range are generally considered acceptable for engineering applications [43].

6

Numerical results on a small DH network

This chapter presents the results obtained by applying the proposed methodology to the first test case, a small DH network. To begin, section 6.2 introduces the test case and its components. Section 6.3 describes the characteristics of the dataset. Next, sections 6.4 and 6.5 discuss the configurations and results of the POD and neural network, respectively. These two sections together establish the ROM used throughout the remainder of the chapter. Then section 6.6 evaluates the prediction performance of the constructed ROM and addresses the second subquestion: “How accurately does the ROM predict the FOM’s state variables, and how much faster can it computes these states for realistic DH networks?”. Finally, section 6.7 examines the impact of integrating the ROM into the optimization. This section answers the third subquestion, “Can using the ROM to generate an initial guess in the optimization yield a speed-up in optimization runtime for such networks?”, and also evaluates the convergence behavior of the results.

6.1. Test setup

The experiments described in this chapter, as well as those presented in chapter 7, were conducted on a machine with the following hardware specifications: AMD EPYC 7763 64-core Processor (4 assigned cores, 8 threads) running at a base clock speed of 2.44 GHz, integrated AMD Radeon Graphics, 32 GB RAM, and Windows 11 Enterprise (version 24H2, 64-bit).

The implementation was carried out in Python (version 3.11.9) using Visual Studio Code.

6.2. DH network architecture

The first test case is considered a small network and contains 3 sources and 217 sinks. An overview of all physical variables and the corresponding number of tracked quantities is provided in table 6.1. A full overview of the components and their quantities can be found in table 6.2. This table also specifies the variables measured for each component and the total number of measurements per component. In addition, it indicates which variables are used as input conditions. Note that pressures at the beginning and end of each pipe segment are distinguished, as the pressure loss is calculated over the section where the fluid is transported through the pipe.

6.3. Snapshot and input generation

The dataset for this test case was derived from historical optimization results. Based on historical input conditions, the optimization process computed what the corresponding state variables would have been, had the digital twin already been in implemented. This optimization was applied to the period from the 8th of December 01:00 to the 23rd of December 00:45, with measurements taken at 15-minute intervals. In total, the dataset comprises 1,440 time intervals.

Table 6.1: Network-wide variable inventory

Variable name	Description	Number of measurements
T_{Fw}	Forward temperature	227
T_{Rt}	Return temperature	227
T_{amb}	Ambient temperature	1
Q	Flow	1,278
P	Power	220
P_{demand}	Estimated power thermal loads	217
p_{Fw}	Forward pressure	227
p_{Rt}	Return pressure	227
$p_{\mathrm{Fw,start}}$	Forward pressure at start component	1,051
$p_{\mathrm{Fw,end}}$	Forward pressure at end component	1,051
$p_{\mathrm{Rt,start}}$	Return pressure at start component	1,051
$p_{\mathrm{Rt,end}}$	Return pressure at end component	1,051
p_{drop}	Pressure drop	13
$p_{\mathrm{drop,Rt}}$	Return-side pressure drop	13

Table 6.2: Component inventory with corresponding variables

Component type	Count	Measured variables		Input conditions	
	Number in network	Type of variables	Total variables for component type	Type of variables	Total inputs for component type
Sinks	217	T_{Fw}, T_{Rt}, Q, P, P_{demand}, p_{Fw}, p_{Rt}	1,519	T_{Rt}, P_{demand}	434
Sources	3	T_{Fw}, T_{Rt}, Q, P, p_{Fw}, p_{Rt}	18	Q, T_{Fw}	6
Pipes	1,051	Q, $p_{\mathrm{Fw,start}}$, $p_{\mathrm{Fw,end}}$, $p_{\mathrm{Rt,start}}$, $p_{\mathrm{Rt,end}}$	5,255	–	–
Shunt	7	T_{Fw}, T_{Rt}, Q	21	–	–
Pumps	12	p_{drop}, $p_{\mathrm{drop,Rt}}$	24	–	–
Valve	1	p_{drop}, $p_{\mathrm{drop,Rt}}$	2	–	–
Weather	1	T_{amb}	1	–	–
Total	**1,292**	**–**	**6,840**	**–**	**440**

During the historical optimization process, all feasible combinations of the decision variables (source forward temperatures) were stored for each time step. Since the neural network is intended to learn the outputs of the network state estimator rather than exclusively the optimal solutions, non-optimal candidates can also serve as valuable data points. This approach expands the dataset to 9,254 instances.

For each candidate, the variables associated with all network components were assembled into a column vector, representing the system's state for that specific instance. Each state vector contains all physical variables of the network (pressures, temperatures, flow rates, and powers) ordered component-wise. The snapshot matrix was then constructed by stacking these column vectors for all candidates. As shown in table 6.2, a total of 6,840 variables are measured, resulting in a snapshot matrix of dimensions 6,840$\times$9,254. Similarly, the input condition matrix was constructed in the same way and has dimensions 440$\times$9,254.

Training, validation and testing set generation

The dataset was divided into three subsets: 70% for training, 15% for validation, and 15% for testing. Allocating the majority to training ensures that the neural network has sufficient examples to learn underlying patterns effectively. Reserving 30% for hyperparameter tuning and final evaluation provides enough data for reliable performance assessment without significantly reducing the training set. This yields a training set of size 6,840$\times$6,478 and test and validation sets of size 6,840$\times$1,388.

The subsets were sampled randomly, meaning consecutive data points within each set do not maintain a temporal relationship. This approach was chosen deliberately because, in the optimization process, a steady-state assumption is applied, making temporal dependencies irrelevant for the model.

6.4. POD

To rewrite the snapshot matrix in a more compact representation, POD is applied to decompose it into output modes and temporal coefficients. This is achieved by performing SVD on the training set. Figure 6.1 illustrates the singular value decay, which shows how the singular values decrease when the mode number increases.

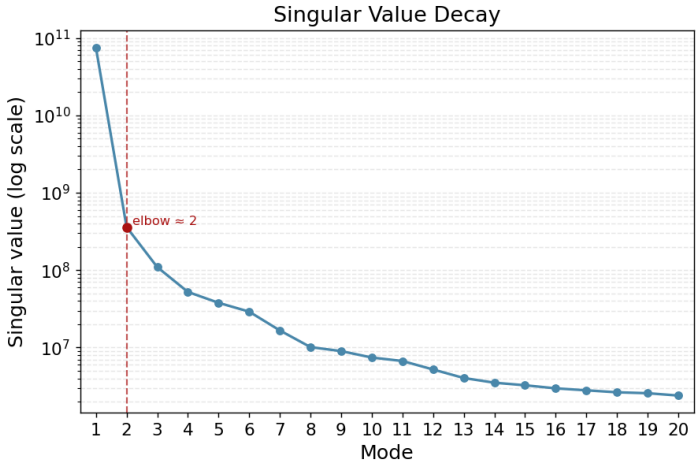


Figure 6.1: Singular value decay of output modes

Each singular value quantifies the amount of energy (or variance) captured by its corresponding mode. Since the singular values are ordered from largest to smallest, the initial modes represent the dominant structures in the data, while later modes capture only minor variations that may be negligible for many applications. A rapid decay therefore indicates that the first few modes contain most of the system's energy.

Let S_k denote the k-th singular value ordered in decreasing magnitude. In the figure, the *elbow* occurs around mode 2. The elbow refers to the point where the curve transitions from steep, i.e. where $S_{k+1}/S_k << 1$, to flat, i.e. where $S_{k+1}/S_k \approx 1$, indicating diminishing returns in energy capture beyond

this mode [16]. In other words, the first two modes explain the majority of the variance, while adding more modes contributes very little additional information. Moreover, the number of modes needed to contain 99% of the energy is only 1, indicating strong dominance in the variation.

While the higher-order modes contribute minimally to the overall energy, they may still contain important fine-scale variations that influence prediction quality. Therefore, the optimal number of POD modes need to be determined by balancing two competing factors: projection error, which decreases as more modes are included, and learnability of the mapping, which typically becomes harder as the dimensionality increases. This trade-off analysis will be addressed in detail in subsection 6.5.

6.5. Neural network

Hyperparameter optimization

The hyperparameters determine the overall structure and behavior of the neural network, and the validation set is used to assess the model's performance across different hyperparameter configurations in order to identify the most effective settings. To ensure that the results do not depend too strongly on any particular train-validation split, we perform four-fold cross-validation. In this study, we do not perform an exhaustive search over all possible parameters, as such an approach would be computationally intensive and falls outside the primary goal of our research. Instead, several elements are held constant while we focus on tuning a selected subset of hyperparameters. Specifically, we fix the loss function as MSE (3.5), employ the Adam optimizer, and use PyTorch's default Kaiming initialization for the network weights and biases. All activation functions considered here are rectified or smooth-rectified nonlinearities for which Kaiming initialization is appropriate.

Our hyperparameter search focuses on identifying the optimal learning rate, activation function, and the depth and width of the feedforward neural network. To this end, we evaluate several architectures that differ both in the number of hidden layers and in the number of neurons per layer. Specifically, we consider the following configurations:

- $[128, 128, 128]$,
- $[64, 128, 256]$,
- $[64, 128, 256, 128, 64]$,
- $[256, 256, 256]$.

These architectures were selected to span a range of model capacities. The inclusion of symmetric structures as well as uniformly wide layers enables us to evaluate whether the system benefits more from gradual feature expansion or uniform layers. Furthermore, deeper and wider networks can be essential for capturing strong nonlinear relationships. However, they can also introduce a greater risk of overfitting. By comparing architectures, we aim to identify the optimal model without compromising generalization.

In addition, we test learning rates of 0.001, 0.0005, and 0.0001 alongside three activation functions; ReLU (3.2), ELU (3.3), and GELU (3.4). We also vary the input dimensionality of the reduced inputs and outputs by using 10, 20 and 50 POD modes. These values correspond to the minimum, intermediate and maximum values of the mode grid examined in subsection 6.5. The purpose here is not to find an optimal number of modes, but rather to assess how changes in input-output dimensionality influence the preferred architecture.

Table 6.3 summarizes the five best performing configurations, ranked by their mean validation loss across folds.

Across the different input-output mode combinations, the average validation loss over four folds was consistently lowest for the architecture with three hidden layers of 256 nodes. To further examine the influence of the learning rate and activation function, figure 6.2 shows the validation loss for the configuration using 50 input and output modes.

From these results, it becomes evident that a learning rate of 0.001 combined with the ELU activation function yields the most favorable performance, as it structurally leads to the lowest validation loss.

Table 6.3: Best performing neural network configurations ranked by average validation loss across four folds.

Rank	Architecture	Learning rate	Activation function	# of modes	Average validation loss
1	$[256, 256, 256]$	0.001	ELU	[50,50]	0.0166
2	$[256, 256, 256]$	0.0005	ELU	[50,50]	0.0203
3	$[256, 256, 256]$	0.0005	RELU	[20,50]	0.0208
4	$[128, 128, 128]$	0.001	ELU	[50,50]	0.0211
5	$[128, 128, 128]$	0.0005	ELU	[20,20]	0.0221

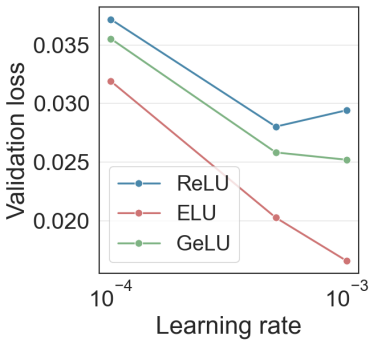


Figure 6.2: Learning rate vs. validation loss for multiple activation functions

POD modes optimization

Determining the optimal number of input and output POD modes is nontrivial, as both choices influence each other and jointly affect model performance. Retaining more output modes generally reduces the output projection error, since a larger number of modes allows the reconstruction to capture finer spatial and temporal variations. However, increasing the output dimensionality also enlarges the neural network's prediction space, which can in turn increase the learning difficulty and the resulting prediction error.

A similar trade-off appears for the input modes. Providing the network with more input modes can expand the information it receives and potentially improve predictability, but only up to a certain point. Beyond this optimum, an excessive number of inputs may overwhelm the model, making it harder to identify the most relevant features and possibly degrading generalization.

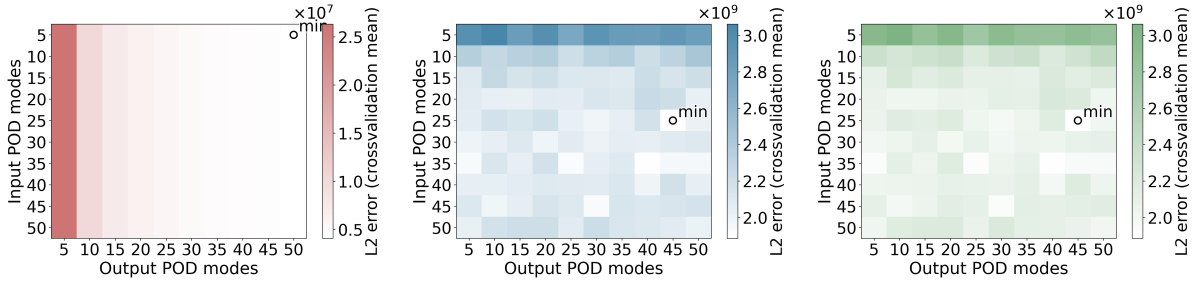
To systematically assess these interactions, we evaluate a range of input-output mode combinations and track three error components: the output projection error (5.2), the prediction error (5.3), and the total error (5.4). For each configuration, the full training process is repeated using four folds for cross-validation to reduce sensitivity to a particular train-validation split. All reported errors correspond to the absolute L_2 error, averaged over the four cross-validation folds.

For the first experiment, the tested values of input and output modes were

$$[5, 10, 15, 20, 25, 30, 35, 40, 45, 50].$$

The corresponding results are shown in figure 6.3. As expected, the output projection error decreases monotonically with the number of output modes. The presence of vertical bands (rather than square patterns) confirm that the input modes have essentially no influence on the output projection error. Figure 6.3a also confirms the diminishing returns of adding more output modes to the projection error; beyond a certain threshold the improvement becomes marginal. The prediction error indicates that very low numbers of input modes yield the highest errors. Remarkably, the structure of the prediction error heatmap closely mirrors that of the total error, implying that the neural network contribution dominates the combined metric.

Because the first grid search indicated that favorable configurations tend to lie toward the upper end of the output-mode range, a second, more refined test was performed. The investigated input-mode



(a) Output projection error **(b)** Prediction error **(c)** Total error

Figure 6.3: Neural network, output projection, and total error for varying input-output mode configurations in the first grid search

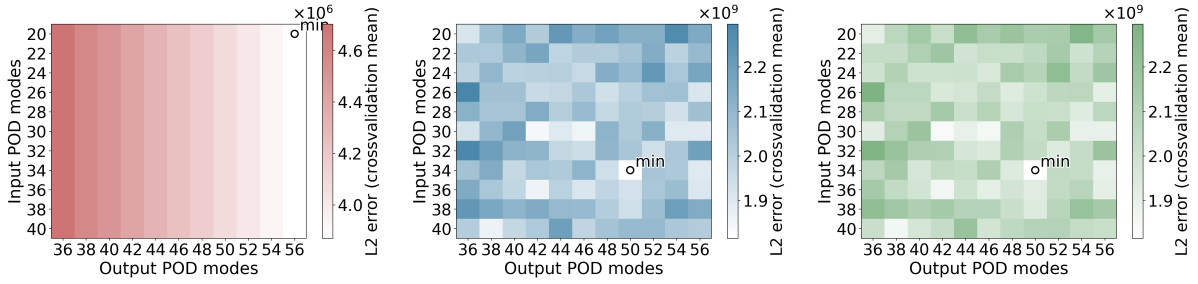
values were

$$[20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40]$$

and the output-mode values

$$[36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56].$$

The results are shown in figure 6.4.



(a) Output projection error **(b)** Prediction error **(c)** Total error

Figure 6.4: Neural network, output projection, and total error for varying input-output mode configurations in the second, refined grid search

The same qualitative trends reappear: increasing the number of output modes consistently reduces the output projection error, and the neural network and total error patterns are again nearly identical. The horizontal striping effect in 6.4b is less pronounced in this refined search, suggesting that in this narrowed input-mode range, changes to the input basis have a greater impact on the network's learning behavior.

Despite the smooth trend visible in the output projection error, the prediction and total error maps display a more irregular structure. This irregular pattern appears because, in this range of POD modes, the dominant part of the system's energy is already captured. This means the true differences between configurations are extremely small, so small effects such as differences between cross-validation folds and randomness from the training process start to dominate the error. As a result, even though some configurations perform slightly better, the effect these different configurations have on the prediction accuracy is minimal.

The output projection and prediction error of all evaluated configurations from both grid searches are summarised in figure 6.5. Each blue dot corresponds to one input-output mode pair.

The figure also illustrates the resulting *Pareto frontier*. In multiobjective optimization, when competing objectives (e.g., low projection error versus low prediction error) cannot be minimized simultaneously. The Pareto set therefore represents the collection of optimal trade-off solutions [32]. Points lying on the Pareto frontier are those configurations for which neither error metric can be reduced without causing an increase in the other. In other words, a configuration is Pareto optimal when no alternative configuration achieves an improvement in one objective without degrading at least one of the others.

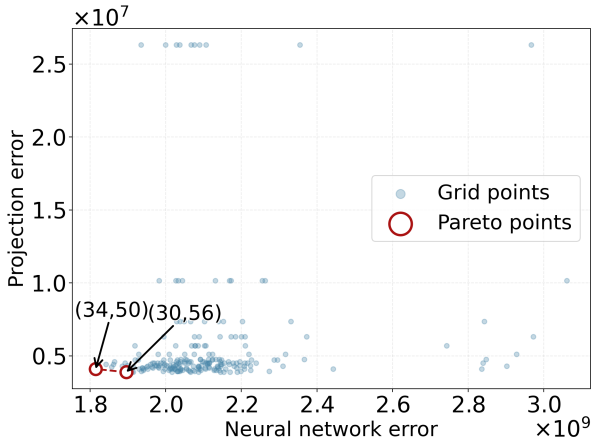


Figure 6.5: Pareto frontier of optimal projection and prediction error

In the present case, the Pareto set contains two configurations: $(34, 50)$ and $(30, 56)$. However, because the prediction error and total error exhibit nearly identical patterns in the heatmaps and because their magnitudes differ by several orders of magnitude, it follows that the neural network contribution has the dominant influence on the total ROM error. Consequently, the configuration with the lower prediction error, namely $(34, 50)$, is selected. This choice is also consistent with the highlighted minima in figures 6.4b (prediction error) and 6.4c (total error).

Interestingly, the optimal configuration lies far beyond what would be suggested by classical POD-based criteria: both the elbow point of the singular value spectrum (2 modes) and the energy-based criterion for 99% captured variance (1 modes) indicate much lower output-mode counts. The input POD shows the same behavior, with the elbow occurring at 7 modes and 99% energy captured with 2 modes. A plausible explanation is that low-energy modes still contain essential nonlinear and temporal features, and discarding them forces the neural network to approximate a far more complex mapping in the reduced space. Increasing the number of input and output modes expands the latent space in which the dynamics are represented, highlighting the fact that increasing both modes significantly improves the learnability of the reduced representation.

The corresponding mean relative L_2 and relative Frobenius input-output projection errors for this configuration are reported in table 6.4.

Table 6.4: Mean relative L_2 and relative Frobenius input and output projection error for training, validation, and test sets for the optimal mode configuration $(34, 50)$

Set	Mean rel. input proj. (%)	Rel. Frob. input proj. (%)	Mean rel. output proj. (%)	Rel. Frob. output proj. (%)
Training	0.87	0.83	$6.15 \cdot 10^{-5}$	$5.96 \cdot 10^{-5}$
Validation	0.88	0.85	$6.30 \cdot 10^{-5}$	$6.09 \cdot 10^{-5}$
Test	0.86	0.83	$6.12 \cdot 10^{-5}$	$5.94 \cdot 10^{-5}$

The extremely low output projection error show that the snapshot matrix lies almost entirely within the subspace spanned by the first 50 POD modes. The small input projection error ($< 1\%$) confirms that the input fields have slightly more variability but are still captured with high accuracy. Furthermore, the errors are nearly identical across train, validation, and test sets, confirming strong generalization.

Results

To improve generalization and prevent overfitting, early stopping was implemented. The patience value was chosen after empirical evaluation. During initial training runs, it was observed that periods of stagnation were often followed by further decreases in loss. A small patience (e.g., 10-20 epochs) frequently stopped training too early, producing models that had higher reconstruction errors when the POD modes were projected back to the physical state variables. Therefore, a patience value of 100 epochs was selected to balance under- and over-training.

The final hyperparameter and architecture settings are summarized in table 6.5.

Table 6.5: Neural network hyperparameters and settings

Asset	Setting/value
Type of neural network	Feedforward
No. hidden layers	3
No. nodes in hidden layers	[256,256,256]
Train/validation/test split	70/15/15
Initialisation	Default (Kaiming)
Activation function	ELU
Loss function	MSE
Optimiser	Adam
Learning rate	0.001
Patience	100
Max no. epochs	10,000
No. input modes	34
No. output modes	50

Figure 6.6 presents the progression of training and validation loss over the course of the epochs. We observe that the losses stagnate around 10^{-2}, indicating that the optimizer has reached a consistent minimum.

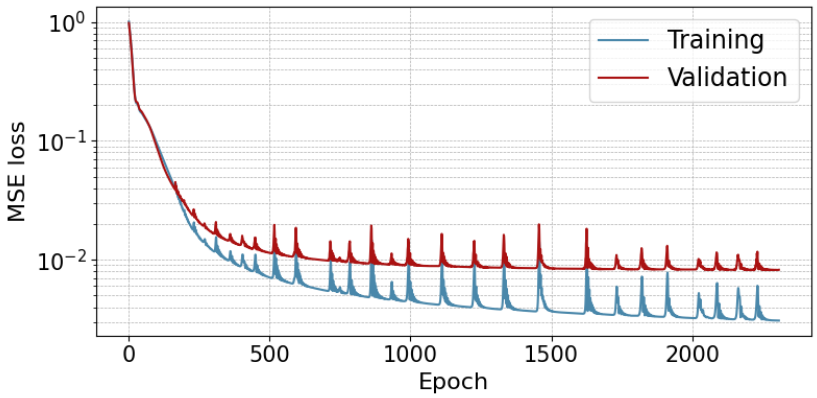


Figure 6.6: Training and validation losses over the training epochs

Detailed measurements of training and prediction times are provided in tables 6.6 and 6.7, respectively. Given that the usual training of a new DH network model is already computationally intensive and typically requires several hours, the additional overhead of approximately 234 seconds ($\approx$ 4 minutes) introduced by the ROM training procedure is negligible. The prediction time is exceptionally small: generating the state variables for 1,388 time steps takes less than one tenth of a second. In comparison, a single iteration of the network state estimator can take up to 100 seconds.

Table 6.6: Breakdown of the total training time for the full workflow (in seconds)

Loading data + input generation	POD on snapshot matrix	POD on input matrix (training and validation)	Training neural network	**Total**
46	77	1	119	**243**

Table 6.7: Timing details of the prediction process (in seconds)

POD on input matrix (test)	Neural network prediction	Reconstruction	**Total**
0.001	0.005	0.028	**0.033**

Table 6.8 contains the mean relative L_2 and relative Frobenius errors of the prediction error (5.3) for the training, validation and test sets.

Table 6.8: Mean relative L_2 and relative Frobenius prediction error for training, validation, and test sets

Set	Mean rel. error (%)	Rel. Frobenius error (%)
Training	0.67	0.72
Validation	0.85	0.94
Test	0.97	0.90

The results show that the neural network achieves consistently low mean relative L_2 and relative Frobenius errors ($< 1\%$) across all data splits, demonstrating that it is sufficiently accurate for predicting the output POD coefficients. The training error is small, and the modest increase ($\approx 0.3\%$) in the validation and test errors indicates that the model is not overfitting and generalizes well to unseen data.

6.6. Prediction performance on the test set

Table 6.9 contains the mean relative L_2 and relative Frobenius error of the total error (5.4) after reconstruction for the training, validation and test set.

Table 6.9: Mean relative L_2 and relative Frobenius total error for training, validation, and test sets

Set	Mean rel. error (%)	Rel. Frobenius error (%)
Training	4.82	0.72
Validation	4.86	0.94
Test	4.79	0.90

The overall total error ($\approx 4.8\%$) remains below the targeted threshold of 5%, and the similarity between the training, validation, and test errors again indicates good generalization. The relative Frobenius errors are noticeably lower than the mean relative L_2 errors, indicating that the prediction error is small when considered globally across the full dataset. The higher mean L_2 values suggest that a few individual variables experience slightly larger local discrepancies, even though the overall reconstruction remains highly accurate. Overall, these results show that the hybrid POD approach provides sufficiently accurate reconstructions for this test case.

Table 6.10: Output projection, prediction and total absolute Frobenius error for training, validation, and test sets

Set	Output proj. Frobenius error	Pred. Frobenius error	Total Frobenius error
Training	$2.13 \cdot 10^{13}$	$2.93 \cdot 10^{17}$	$2.93 \cdot 10^{17}$
Validation	$4.79 \cdot 10^{12}$	$1.07 \cdot 10^{17}$	$1.07 \cdot 10^{17}$
Test	$4.51 \cdot 10^{12}$	$9.73 \cdot 10^{16}$	$9.73 \cdot 10^{16}$

Table 6.10 presents the output projection, prediction, and total absolute Frobenius errors for the training, validation, and test sets. The absolute values appear extremely large, but this is a consequence of the problem scale. The output consist of 6,480 variables, of which a subset are power-related quantities with magnitudes up to 10^5. Because the Frobenius norm aggregates squared errors over all variables, the resulting totals reach $10^{16} - 10^{17}$ without necessarily implying poor predictive performance.

Although the total Frobenius error is the sum of the output projection and prediction error, this is not immediately visible because the two terms differ by several orders of magnitude. The projection error lies around $10^{12} - 10^{13}$, whereas the prediction error is around 10^{17}. As a result, the total error is numerically almost identical to the prediction error. This confirms that the prediction component overwhelmingly dominates the total ROM error. This observation is consistent with the findings from the mode optimization and confirms that further improvements in ROM accuracy are more likely to come from better coefficient prediction rather than from expanding or refining the POD basis.

6.7. Performance in optimization

Convergence

The optimization is performed over a time period ranging from the 3rd of December 00:00 to the 4th of December 02:00, resulting in a data set with 104 distinct instances. As explained in section 2.4, the state iteration of a candidate is considered converged when the convergence criterion is satisfied for every time step individually. The optimization procedure continues updating candidates until the predetermined maximum number of candidates is reached. In this test case, the number of candidates was set to 13. For two of the evaluated time steps, no feasible solution was found.

The amount of iterations needed for each candidate to converge is given in table 6.11.

Table 6.11: Amount of iterations of state iteration per candidate for both configurations

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
With ROM	5	7	6	5	6	4	4	4	4	3	3	2	3
Without ROM	5	7	6	5	6	5	5	5	5	5	5	5	5

It becomes clear that from candidate 5 onward, the configuration including the ROM reduces the number of iterations by one, and from candidate 9 onwards even by two or three.

For this test case, two independent convergence criteria are evaluated at every time step: one for the sink powers and one for the shunt flows. A time step is considered converged only if both criteria fall below their respective tolerances. A candidate is considered converged once all time steps satisfy both criteria.

The first criterion evaluates how consistent the iterated sink powers are with the initial guess. Each time step contains 217 sinks, and the iteration updates their power values until they satisfy the hydraulic and thermal balance. To quantify this mismatch, the total absolute error at time step t is defined as

$$\text{Total Absolute Error (TAE)}(t) = \sum_{i=1}^{217} \left| P_{i,\text{guess}}(t) - P_{i,\text{iter}}(t) \right|,$$

where $P_{i,\text{guess}}(t)$ is the initial guessed power at component i and time step t, and $P_{i,\text{iter}}(t)$ is the value obtained after one iteration. The sink-power deviation is considered acceptable when

$$\text{TAE}(t) \leq \frac{\epsilon \cdot P_{\text{total}}}{100} = \text{tol}_{\text{sink}}(t), \tag{6.1}$$

where ϵ is a predetermined tolerance and P_{total} is the total sink demand. Since P_{total} depends on the decision variables, the tolerance varies per candidate and per time step.

The second criterion assesses the consistency the shunt flows. There are 7 shunts, indexed by j. As with the sink powers, the iteration updates the shunt flows until they are consistent with the hydraulic state. The total absolute error for a single time step is defined as

$$\text{Total Absolute Error (TAE)}(t) = \sum_{j=1}^{7} \left| Q_{j,\text{guess}}(t) - Q_{j,\text{iter}}(t) \right|,$$

where $Q_{i,\text{guess}}(t)$ is the initial guessed flow at component i and time step t, and $Q_{i,\text{iter}}(t)$ is the flow estimated by the iteration at that same time step. This measures the deviation between the guessed and iterated shunt flows across all shunts.

The shunt-flow deviation is considered acceptable when

$$\text{TAE}(t) \leq \frac{\epsilon \cdot Q_{\text{total}}}{100} = \text{tol}_{\text{shunt}}(t), \tag{6.2}$$

where ϵ is a predetermined tolerance and Q_{total} denotes the total supply of all sources combined.

In this test case, ϵ was set to 1, corresponding to a maximum allowed deviation of 1%. A candidate is considered converged if (6.1) and (6.2) hold for all time steps t simultaneously.

To gain more insight into how and when a candidate converges, figure 6.7 shows the evolution of the two convergence criteria for both the configuration with ROM and without ROM. Each line corresponds to one optimization candidate. Darker lines represent earlier candidates; lighter lines represent later candidates. The vertical axis shows the max ratio, defined as

$$\text{max ratio} = \max_t \left(\frac{\text{TAE}(t)}{\text{tol}(t)} \right).$$

The max ratio normalizes the total absolute error by the allowed deviation at each time step. In essence, the max ratio represents the worst-performing time step of a candidate. A candidate is considered converged once its max ratio is below 1.

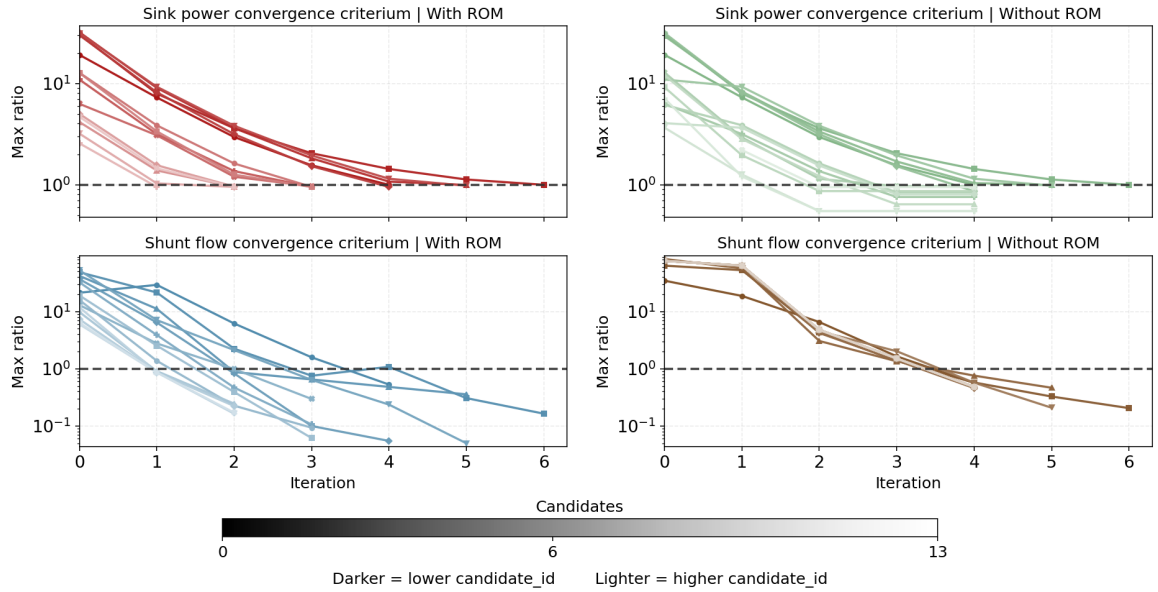


Figure 6.7: Convergence trajectories of the sink power and shunt flow criteria for all optimization candidates (referred to as candidate id). The plots compare the ROM-based (left panels) with the FOM optimization (right panels)

The curves decrease because each iteration updates the guessed state, reducing the mismatch between guessed and iterated values. Once the max ratio drops below 1, the criterion is satisfied for all time steps.

We observe that, for the configuration without the ROM, the shunt flow criterion was the decisive factor. Although the sink powers had already converged for most candidates, the state iteration continued until both criteria were satisfied. Remarkably, this behavior is reversed in the configuration with ROM: in that case, the sink power criterion requires more iterations to converge than the shunt flow criterion for all candidates. This indicates that the ROM affects the optimality of the sink powers negatively, making their convergence slightly more demanding, while it improves the convergence of the shunt flows.

However, this metric only reflects the progress of the ”worst” time step. It is therefore possible that the iteration continues solely because a small subset of time steps has not yet converged. For this reason, it is useful to examine the distribution of the convergence evolution across all time steps.

Figures 6.8, 6.9 and 6.10 show the distribution of the convergence ratios across all time steps for candidates 0, 6, and 12, respectively. Corresponding plots for all evaluated candidates are provided in appendix B.

The ratio is defined as

$$\text{ratio} = \frac{\text{TAE}(t)}{\text{tol}(t)},$$

where ϵ denotes the predefined tolerance. The solid line represents the median, the darker shaded band the 25th-75th percentile, and the lighter outer band the 10th-90th percentile.

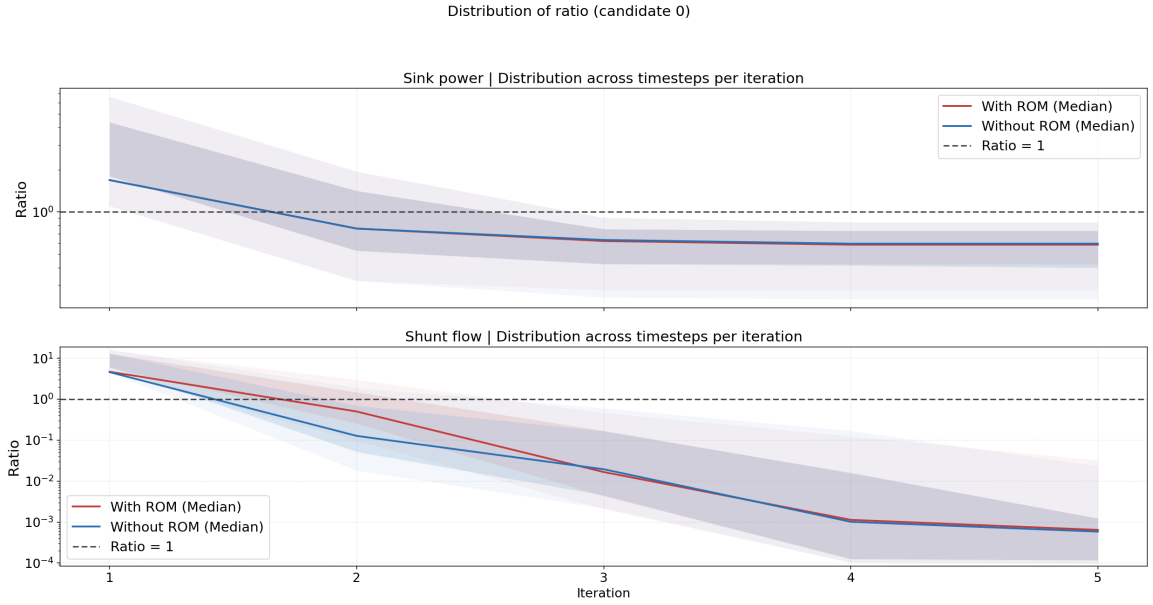


Figure 6.8: Distribution of the convergence ratio for the sink power (top) and shunt flow (bottom) criterion across all time steps per iteration for candidate 0. Solid lines show the median ratio per iteration for the simulations with and without the ROM, while the shaded regions indicate the spread across time steps.

For candidate 0, the median ratio of the sink power criterion decreases at a similar rate for both the ROM and non-ROM simulations. The distribution is initially wide but narrows considerably from iteration 2 onward, and the two median curves lie almost exactly on top of each other, indicating essentially identical convergence behavior. For the shunt flow criterion, the ratios decrease much more rapidly than for sink power, falling below the tolerance threshold within two to three iterations. The ROM run shows a slightly faster reduction in the initial iterations, but both approaches converge to nearly the same final level.

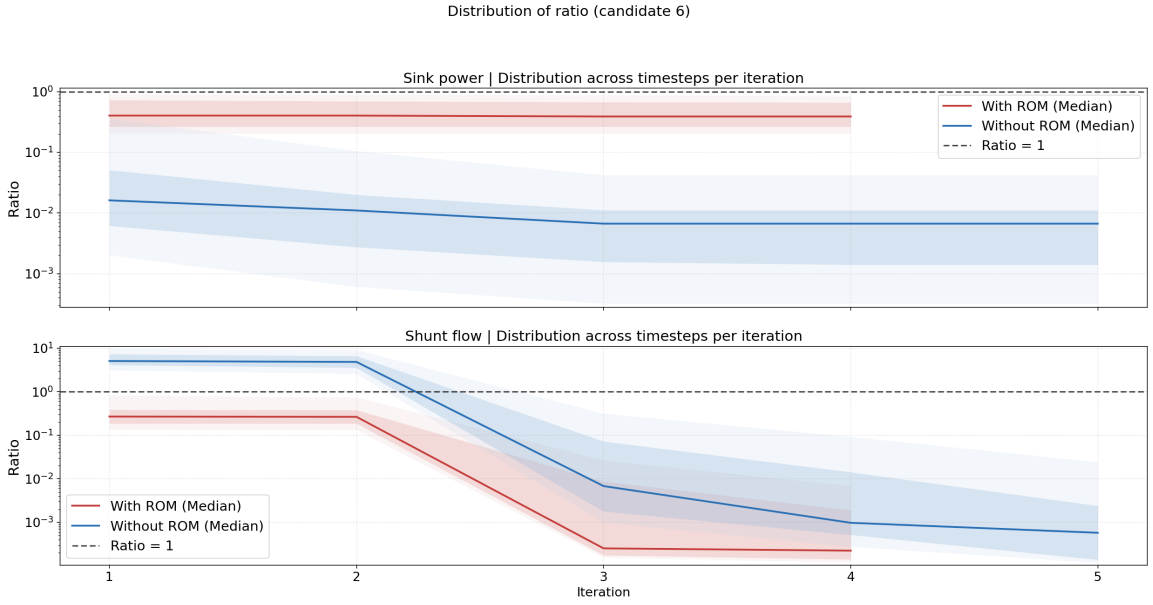


Figure 6.9: Distribution of the convergence ratio for the sink power (top) and shunt flow (bottom) criterion across all time steps per iteration for candidate 6. Solid lines show the median ratio per iteration for the simulations with and without the ROM, while the shaded regions indicate the spread across time steps.

For candidate 6, the median ratio for the sink power criterion is noticeably lower for the non-ROM simulation, and its distribution is also much more spread out, indicating larger variability across time steps. In contrast, the ROM median remains closer to the tolerance boundary at ratio = 1, and its distribution is more compact. Since the ROM run terminates at iteration 4 while the non-ROM run continues to iteration 5, this suggests that fewer than 10% of the non-ROM time steps have not yet converged at iteration 4, whereas the ROM is able to reduce these remaining cases sufficiently earlier.

For the shunt flow criterion, most of the reduction occurs within the first three iterations for both runs. Again, the earlier stopping of the ROM indicates that a small subset of non-ROM time steps remains above tolerance at iteration 4. The ROM appears to handle these more challenging time steps more effectively, allowing it to reach the convergence threshold sooner.

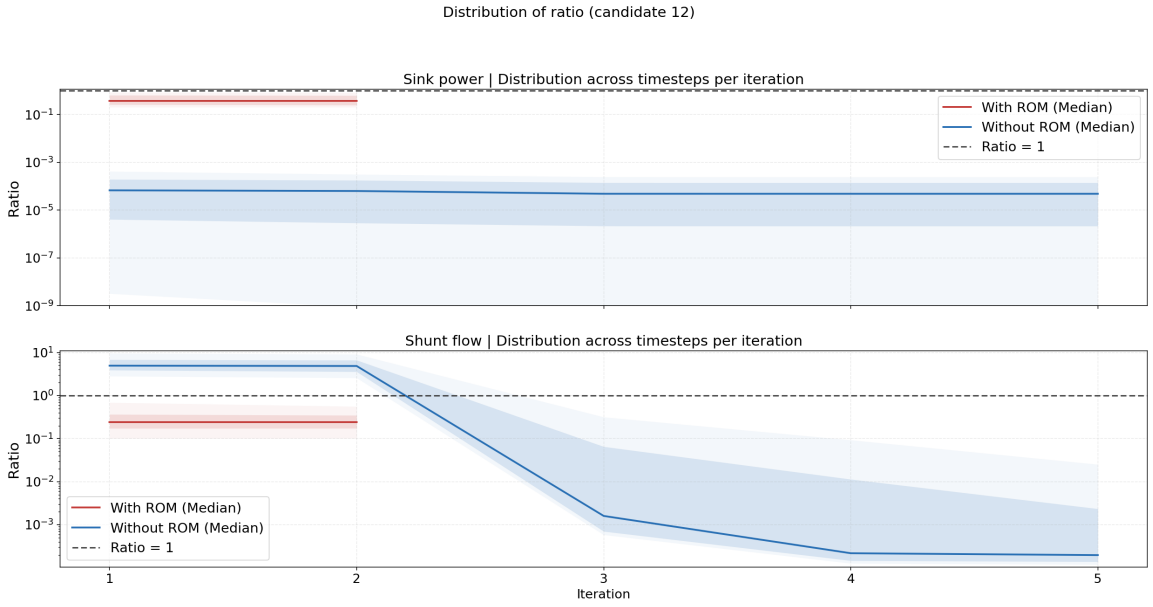


Figure 6.10: Distribution of the convergence ratio for the sink power (top) and shunt flow (bottom) criterion across all time steps per iteration for candidate 12. Solid lines show the median ratio per iteration for the simulations with and without the ROM, while the shaded regions indicate the spread across time steps.

For candidate 12, both the sink power and shunt flow criteria show no decrease in the median ratio or in the spread when using the ROM. In both cases, the ROM curves lie consistently below the ratio $= 1$ threshold, indicating that the vast majority of time steps were already converged after the first iteration and that the second iteration succeeded in bringing all remaining time steps below the tolerance. In other words, the ROM completes convergence almost immediately.

In contrast, the non-ROM simulation displays different behavior for the two criteria. For sink power, the reduction in the median ratio is slow and gradual, with a wide distribution indicating variability across time steps. For shunt flow, the reduction is much faster: the median ratio enters the tolerance region within roughly three iterations. However, the final 10% (or fewer) of time steps remain above tolerance for longer, which forces the non-ROM solver to continue iterating even though most time steps have already converged.

Overall, the ROM and non-ROM configurations behave similarly for the early candidates, but clear differences emerge from candidate 6 onward. In these later cases, the ROM typically converges in fewer iterations and shows a tighter distribution across time steps, meaning that fewer time steps require updates. This reduces both the number of iterations and the computational work per iteration.

This behavior can be understood from how close each candidate starts to the converged state. The early candidates begin relatively far from the final solution, so both the ROM and the FOM must perform a similar number of corrective steps, resulting in comparable convergence patterns. From candidate 6 onward, however, the initial state is much closer to the optimum, and the ROM provides a better initial

guess across time steps. As a result, the ROM reduces the number of time steps that remain above tolerance after each iteration, allowing convergence to be reached earlier.

For candidate 12, the ROM starts almost at the converged solution and satisfies both criteria almost immediately, whereas the non-ROM solver continues because a small subset of time steps remains above the tolerance threshold. The combined effect of the ROM, skipping the first FOM iteration, converging in fewer total iterations, and operating on fewer time steps per iteration, ultimately accounts for the overall speed-up of the ROM approach.

Results

Table 6.12 presents the runtimes of the optimization procedure with and without the ROM. The state iteration time reflects only the cost of computing the system states across all iterations, which is the part directly accelerated by the ROM. The total optimization time additionally includes all additional operations performed by the optimizer, such as checking whether new iterates improve the objective, updating time steps when necessary and performing convergence checks. These tasks are unaffected by the ROM.

Table 6.12: Comparison of state iteration and optimization runtimes with and without the ROM

	State iteration time (s)	Total optimization time (s)
With ROM	957.50	1202.42
Without ROM	1119.86	1401.50

Using the results in table 6.12, we observe that incorporating the ROM yields a speed-up factor (3.9) of $\times$ 1.17 for the state iteration. This reduction ultimately translates to an overall optimization speed-up factor of 1.17.

To assess whether the optimization yields the same solution when using the ROM, the forward temperature trajectories of the sources are compared in figure 6.11. The dashed lines correspond to the ROM-based optimization, while the solid lines show the results obtained without the ROM. The source forward temperatures are examined because they represent the decision variables of the optimization problem.

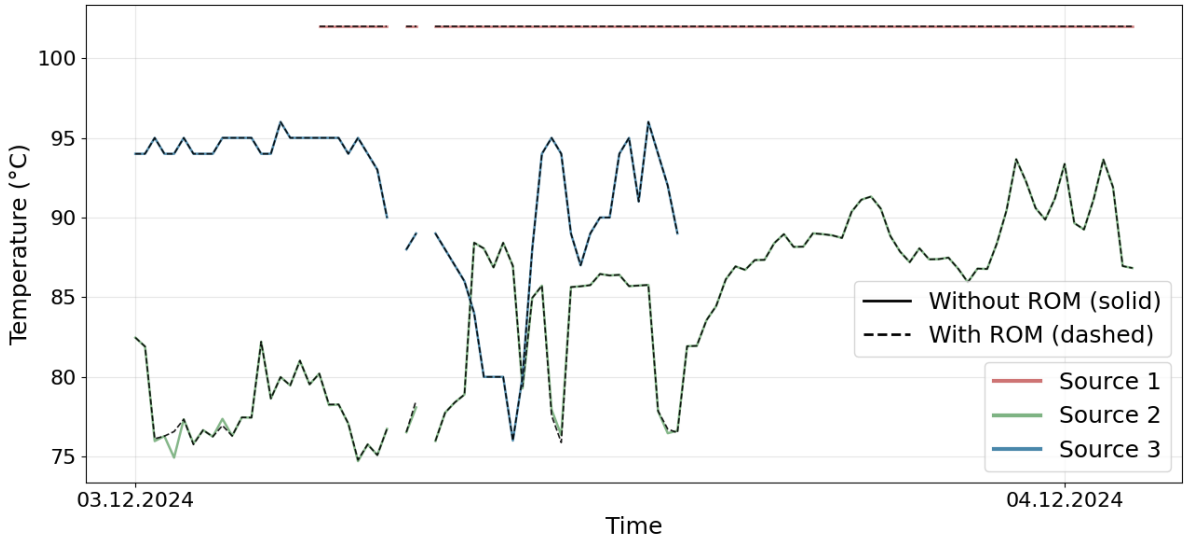


Figure 6.11: Comparison of source forward temperature results for the optimization with and without the ROM

The two configurations produce nearly identical trajectories. Only a very small deviation is visible around 03.12 01:00 for source 2, corresponding to a relative L_2 error of 0.05% between the two solutions. For the remaining sources, the trajectories coincide exactly, resulting in an error of 0.00%. Overall, this leads to a mean relative L_2 error across all sources of approximately 0.02%.

The two visible gaps in the trajectories correspond to moments where no feasible solution was found for the corresponding time steps. In addition, the results show that sources 1 and 3 are not operated continuously throughout the period. This indicates that source 2 functions as the primary heat supplier, while the other two sources are activated only when additional energy production is required.

6.8. Conclusion

This chapter set out to investigate how the ROM performs when integrated into the state iteration procedure, specifically in terms of accuracy, computational efficiency, and its potential to accelerate the overall optimization process. The evaluation showed that the total mean relative L_2 error (5.4) on the test set was 4.79%, remaining below the target threshold of 5%, which indicates that the ROM achieves the required level of accuracy. In terms of computational performance, the ROM required only 0.033 seconds to generate a prediction, whereas a single iteration of the full state iteration procedure can take up to 100 seconds. Training the ROM for this test case took approximately 4 minutes.

Furthermore, when the ROM was incorporated into the optimization loop, it achieved a speed-up of $\times$ 1.17, demonstrating that for this test case the objective of accelerating the optimization was successfully met. Investigating the convergence per candidate shows that the ROM's speed-up arises from skipping the first FOM iteration, converging in fewer total iterations, and requiring updates for fewer time steps per iteration. Having validated the method on a small DH network, the next step is to extend the approach to a larger DH network to assess its scalability.

7

Numerical results on a large DH network

This chapter presents the results of applying the proposed methodology to a large-scale DH network. Section 7.1 first outlines the test case and its main components, followed by section 7.2, which examines the characteristics of the available dataset. Sections 7.3 and 7.4 then detail the configurations and outcomes of the POD and neural network. Finally, section 7.5 evaluates the predictive performance of the resulting ROM, with specific attention to the forward temperatures at the source nodes.

7.1. DH network architecture

The larger test case consists of three sources and 397 sinks, nearly three times as many sinks as in the previous test case. An overview of all physical variables and the corresponding number of tracked quantities is provided in table 7.1. The components included in this network, along with their quantities, are summarized in table 7.2, which also lists the monitored variables and the total number of measurements. In contrast to the previous test case, this DH network contains no valves, pumps, or shunts.

Table 7.1: Network-wide variable inventory

Variable name	Description	Number of measurements
T_{Fw}	Forward temperature	2,782
T_{Rt}	Return temperature	2,782
T_{amb}	Ambient temperature	1
Q	Flow	1,878
P	Power	2,779
P_{demand}	Estimated power thermal loads	2,779
p_{Fw}	Forward pressure	2,782
p_{Rt}	Return pressure	2,782
$p_{\mathrm{Fw,start}}$	Forward pressure at start component	7,390
$p_{\mathrm{Fw,end}}$	Forward pressure at end component	7,390
$p_{\mathrm{Rt,start}}$	Return pressure at start component	7,390
$p_{\mathrm{Rt,end}}$	Return pressure at end component	7,390

7.2. Snapshot and input generation

The snapshot data extracted from the historical optimization covers the period from the 1st of February 01:00 to the 28th of February 00:45, with measurements taken at 15-minute intervals. Compared to

Table 7.2: Component inventory with corresponding variables

Component type	Count	Measured variables		Input conditions	
	Number in network	Type of variables	Total variables for component type	Type of variables	Total inputs for component type
Sinks	397	T_{Fw}, T_{Rt}, Q, P, P_{demand}, p_{Fw}, p_{Rt}	2,779	T_{Rt}, P_{demand}	794
Sources	3	T_{Fw}, T_{Rt}, Q, P, p_{Fw}, p_{Rt}	18	Q, T_{Fw}	6
Pipes	1,478	Q, $p_{\mathrm{Fw,start}}$, $p_{\mathrm{Fw,end}}$, $p_{\mathrm{Rt,start}}$, $p_{\mathrm{Rt,end}}$	7,390	–	–
Weather	1	T_{amb}	1	–	–
Total	**1,878**	**–**	**10,188**	**–**	**800**

the previous test case, this represents an extension of 13 additional days.

In total, the dataset contains 2,592 time intervals, which expands to 25,883 snapshots when all potential candidates are included. As shown in table 7.2, a total of 10,188 variables are monitored, yielding a snapshot matrix of size 10,188×25,883. Similarly, the input condition matrix was constructed using the same structure and has dimensions 800×25,883.

Training, validation and testing set generation

The dataset was again divided into three subsets: 70% for training, 15% for validation, and 15% for testing, with all samples assigned randomly. This results in a training set of size 10,188×18,118 and validation and test sets each of size 10,188×3,882.

7.3. POD

To decompose this high-dimensional matrix into a linear combination of reduced bases, POD is applied to the training set using SVD, yielding both the output modes and their corresponding temporal coefficients. The singular value decay of the output modes is shown in figure 7.1.

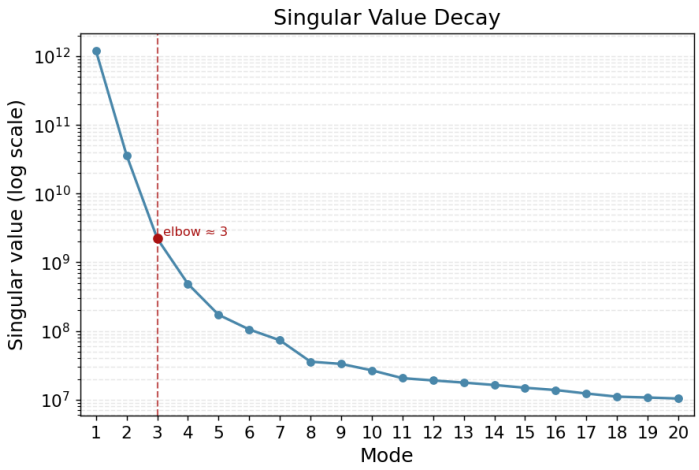


Figure 7.1: Singular value decay

Compared to the previous test case, the elbow in the singular value spectrum shifts from 2 to 3. This is expected, as the number of variables represented in the POD decomposition has increased by a factor of approximately 1.5. Nevertheless, the dominant dynamics of the system are still captured effectively by the leading POD modes. Notably, retaining only a single mode again already preserves 99% of the total energy.

The final number of POD modes used in the ROM, however, also depends on the learnability of the

neural network. A detailed analysis of this aspect is presented in section 7.4.

7.4. Neural network

Hyperparameter optimization

The hyperparameter optimization setup is identical to the one used in the small test case (section 6.5). Table 7.3 summarizes the five best performing configurations, ranked by their mean validation loss across folds.

Table 7.3: Best performing neural network configurations ranked by average validation loss across four folds.

Rank	Architecture	Learning rate	Activation function	# of modes	Average validation loss
1	$[256, 256, 256]$	0.001	GELU	[50,50]	0.0286
2	$[256, 256, 256]$	0.001	GELU	[20,50]	0.0293
3	$[128, 128, 128]$	0.001	GELU	[50,50]	0.0295
4	$[128, 128, 128]$	0.001	ELU	[50,50]	0.0296
5	$[256, 256, 256]$	0.001	ELU	[50,20]	0.0300

As before, the average validation loss across four folds was consistently lowest for the $[256, 256, 256]$ architecture. Figure 7.2 illustrates the relationship between validation loss, activation function, and learning rate for the case with 50 input and output modes. It becomes apparent that a learning rate of 0.001 combined with the GELU activation function yields the lowest validation loss. Since the optimal learning rate and network depth/width are the same as in the previous test case, the only hyperparameter that meaningfully differentiates in this larger system is the activation function.

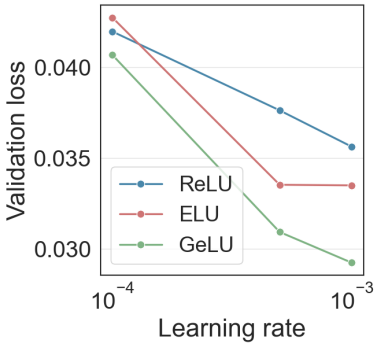


Figure 7.2: Learning rate vs. validation loss for multiple activation functions

Mode optimization

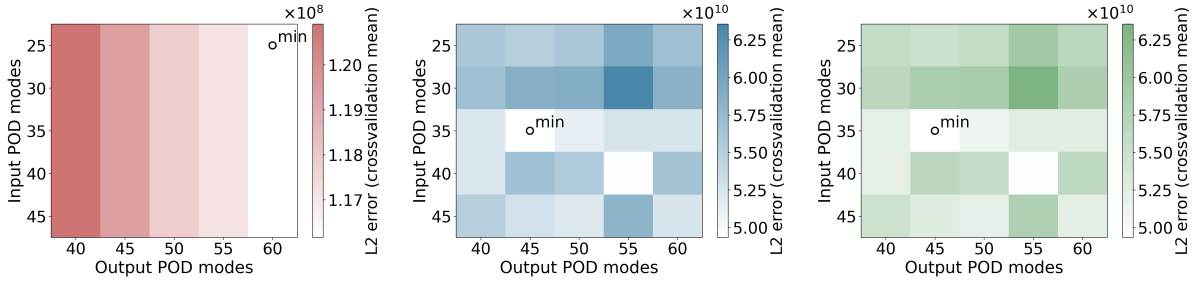
To assess the optimal number of input and output POD modes, a range of input-output mode combinations is evaluated while tracking the output projection error (5.2), prediction error (5.3), and total error (5.4). The first test case indicated that optimal performance was obtained for configurations with input mode counts above approximately 25 and output mode counts above 40. Based on this insight, the present test case focuses on fewer options but with a higher range of mode values. Therefore, the input-mode values investigated are

$$[25, 30, 35, 40, 45]$$

and the output-modes values are

$$[40, 45, 50, 55, 60].$$

The corresponding results are presented in figure 7.3. As before, it is evident that increasing the number of output modes consistently reduces the output projection error. Since the initial grid search suggested that favorable configurations tend to cluster toward the lower end of the output-mode range, a second,



(a) Output projection error **(b)** Prediction error **(c)** Total error

Figure 7.3: Neural network, output projection, and total error for varying input-output mode configurations in the first grid search

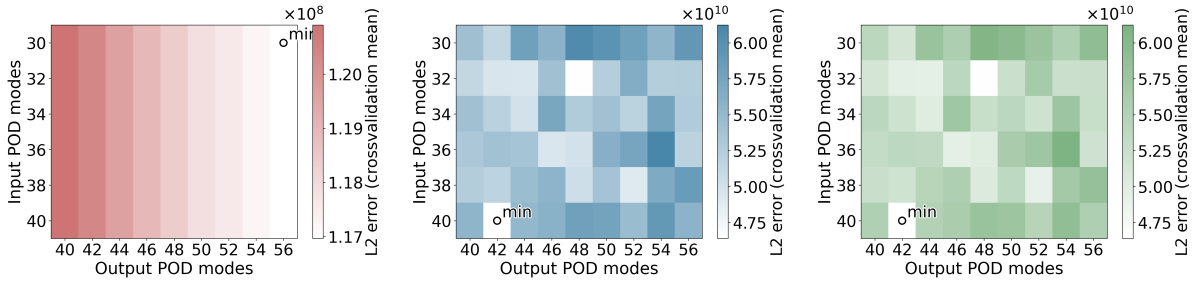
more refined experiment was conducted. In this refined search, the input-mode values investigated were

$$[30, 32, 34, 36, 38, 40]$$

and the output-mode values were

$$[40, 42, 44, 46, 48, 50, 52, 54, 56].$$

The corresponding results are shown in figure 7.4.



(a) Output projection error **(b)** Prediction error **(c)** Total error

Figure 7.4: Neural network, output projection, and total error for varying input-output mode configurations in the second, refined grid search

The optimal configuration appears to be 40 input modes and 42 output modes, which is again significantly higher than the elbow point (6 modes for input, 3 modes for output) and the energy-based criterion of 99% (1 mode for input and output).

The projection and prediction errors for all evaluated configurations from both grid searches are summarized in figure 7.5. Each blue dot in this figure corresponds to one specific input-output mode pair and the red dots correspond to the Pareto frontier.

The Pareto set contains those configurations for which neither error metric can be reduced without increasing the other. Interestingly, the Pareto set for this test case is substantially larger than in the previous one.

This figure also reinforces the pattern observed earlier: the prediction error plays a much more dominant role in determining the total error. In figure 7.5, this is reflected by the fact that configurations with low prediction error correspond to higher output projection error. When examining the total error in figure 7.4c, it becomes evident that the configuration with 40 input modes and 42 output modes is optimal, even though that leads to the highest output projection error in the Pareto set.

The corresponding input and output projection errors for this configuration are summarized in table 7.4.

Compared to the small test case, a slight increase can be seen in the input projection errors. Nevertheless, these values remain close to 1% across all datasets and show almost no variation between training, validation, and test sets. The output projection errors are again extremely small.

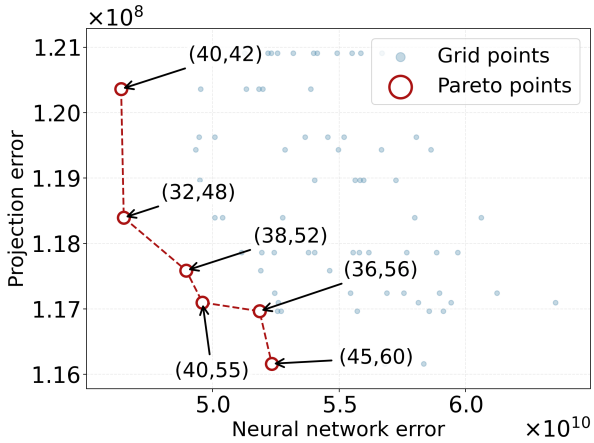


Figure 7.5: Pareto frontier of optimal projection and prediction error

Table 7.4: Mean relative L_2 and relative Frobenius input and output projection error for training, validation, and test sets for the optimal mode configuration $(40, 42)$

Set	Mean rel. input proj. (%)	Rel. Frob. input proj. (%)	Mean rel. output proj. (%)	Rel. Frob. output proj. (%)
Training	1.19	1.19	$1.80 \cdot 10^{-5}$	$1.82 \cdot 10^{-5}$
Validation	1.20	1.20	$1.82 \cdot 10^{-5}$	$1.83 \cdot 10^{-5}$
Test	1.20	1.20	$1.81 \cdot 10^{-5}$	$1.82 \cdot 10^{-5}$

Results

As before, early stopping with a patience of 100 epochs is applied. The final hyperparameter and architecture settings are summarized in table 7.5.

Table 7.5: Neural network hyperparameters and settings

Asset	Setting/value
Type of neural network	Feedforward
No. hidden layers	3
No. nodes in hidden layers	[256,256,256]
Train/validation/test split	70/15/15
Initialization	Default (Kaiming)
Activation function	GELU
Loss function	MSE
Optimiser	Adam
Learning rate	0.001
Patience	100
Max no. epochs	10,000
No. input modes	40
No. output modes	42

Compared to the previous test case, the only settings that differ are the activation function and optimal amount of input and output modes.

Figure 7.6 shows the evolution of the training and validation losses over the epochs. As both losses continue to decrease and eventually stagnate around 10^{-1}, we conclude that the optimizer has reached a consistent minimum. Furthermore, the fact that the validation loss remains close to the training loss indicates that the additional data likely helped mitigate overfitting.

Timing details for both training and prediction are reported in tables 7.6 and 7.7, respectively. As expected, the training time increases substantially compared to the small test case. This is not surprising,

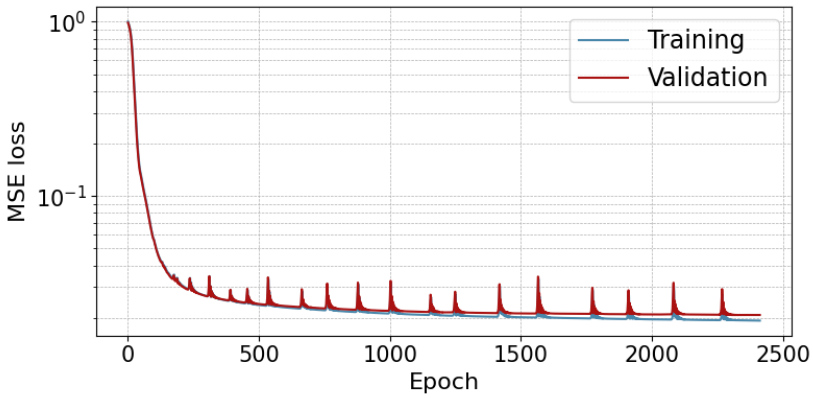


Figure 7.6: Training and validation losses over the training epochs

given that the present dataset contains more than two and a half times as many samples, and the network architecture is also larger. Nevertheless, the training process can be integrated into existing workflows and does not impact the real-time applicability of the ROM. The prediction time, on the other hand, remains almost negligible.

Table 7.6: Breakdown of the total training time for the full workflow (in seconds)

Loading data + input generation	POD on snapshot matrix	POD on input matrix (training and validation)	Training neural network	**Total**
242	679	2	325	**1248**

Table 7.7: Timing details of the prediction process (in seconds)

POD on input matrix (test)	Neural network prediction	Reconstruction	**Total**
0.003	0.009	0.073	**0.085**

Table 7.8 contains the mean relative L_2 and relative Frobenius error of the prediction (5.3) for the training, validation and test set. The mean relative L_2 errors for the training, validation, and test sets

Table 7.8: Mean relative L_2 and relative Frobenius prediction error for training, validation, and test sets

Set	**Mean rel. error (%)**	**Rel. Frobenius error (%)**
Training	0.96	1.77
Validation	1.01	1.83
Test	0.99	1.88

are all close to 1%, with only minimal variation between them. This consistency indicates that the neural network has learned an input-output mapping without overfitting. The validation error being slightly higher than the training error is expected, while the test error falling between the two confirms good generalization performance. For this case, the relative Frobenius errors are higher than the mean relative L_2 errors, meaning that although individual coefficients have small local errors, these accumulate coherently across the full dataset, resulting in a larger global error.

7.5. Prediction performance on the test set

Table 7.9 contains the mean relative L_2 and relative Frobenius error of the total error (5.4) after reconstruction for the training, validation and test set.

The total error of $\approx$ 3.6 % remains well below the desired threshold of 5%, and is even 1.2% lower than that of the previous test case. In contrast to the prediction error, the total mean relative L_2 errors are

Table 7.9: Mean relative L_2 and relative Frobenius total error for training, validation, and test sets

Set	Mean rel. error (%)	Rel. Frobenius error (%)
Training	3.63	1.77
Validation	3.61	1.77
Test	3.62	1.88

higher than the Frobenius errors. This indicates that a few variables exhibit larger localized discrepancies, raising the per-variable average, while the global Frobenius norm remains lower because most variables are reconstructed accurately.

Table 7.10: Output projection, prediction and total absolute Frobenius error for training, validation, and test sets

Set	Output proj. Frobenius error	Pred. Frobenius error	Total Frobenius error
Training	$4.81 \cdot 10^{14}$	$4.56 \cdot 10^{20}$	$4.56 \cdot 10^{20}$
Validation	$1.04 \cdot 10^{14}$	$1.04 \cdot 10^{20}$	$1.04 \cdot 10^{20}$
Test	$1.03 \cdot 10^{14}$	$1.10 \cdot 10^{20}$	$1.10 \cdot 10^{20}$

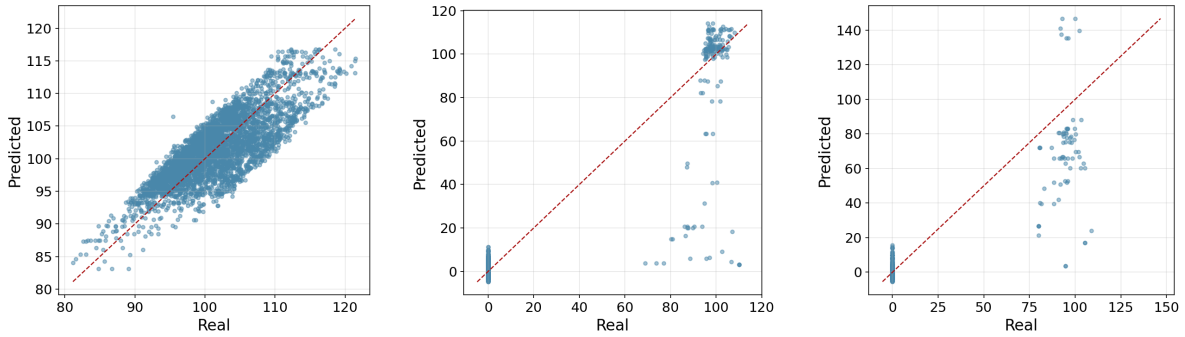
Table 7.10 contains the output projection, prediction and total absolute Frobenius error for the training, validation and test set. As in the previous test case, the total error is effectively identical to the prediction error. The projection error therefore contributes negligibly to the overall reconstruction error, which is again dominated by the neural network prediction.

Overall, these results indicate that the hybrid POD-ROM achieves a sufficiently accurate reconstruction for this larger network as well.

7.5.1. Prediction quality of forward temperatures

During the integration of the ROM for the larger test case within the optimization framework, it became apparent that no meaningful optimal values were obtained for source 3, and the results for sources 1 and 2 deviated noticeably from the FOM reference. A closer inspection of the neural network predictions revealed that the model occasionally produced negative values for both power and temperature at sources 2 and 3. In addition, the predicted forward temperatures T_{Fw} were frequently unrealistically low, often below $20\,^{\circ}\mathrm{C}$, which is physically impossible in DH networks. Figure 7.7 shows the predicted versus true values of T_{Fw} for the three sources. The discrepancy is especially pronounced for sources 2 and 3. These results indicate not only a lack of physical consistency but also violations of fundamental operational constraints.

The vast majority of variables are predicted accurately, so their small errors dominate the mean relative error. The problematic source temperatures therefore contribute only marginally to the aggregated metric, even though they represent severe local inconsistencies that can propagate through the ROM and violate physical constraints. Consequently, even if the model displays a low mean relative error, individual outliers can significantly affect downstream components of the computational pipeline.



(a) Source 1 **(b)** Source 2 **(c)** Source 3

Figure 7.7: Predicted versus true values of T_{Fw} for the three sources.

To better understand the origin of these issues, the training data for T_{Fw} were inspected, as shown in

figure 7.8. A clear pattern emerges: source 1 operates continuously throughout the dataset, whereas sources 2 and 3 are active only during short periods.

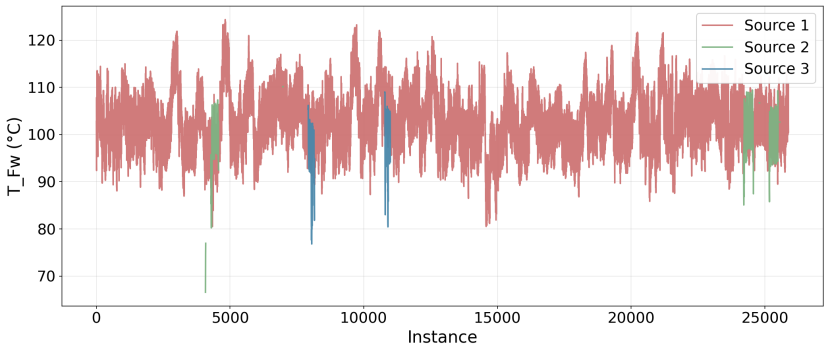


Figure 7.8: Training time series of T_{Fw} for the three sources

For all timestamps in which sources 2 and 3 were switched off, the recorded power was zero and the corresponding temperatures are irrelevant when the source is not producing heat. Consequently, the dataset contains empty entries for these periods, which were filled with zeros during preprocessing.

By replacing the empty entries with zeros, the preprocessing step therefore introduced a physically implausible pattern into the data. The neural network learned from this that near-zero temperatures were typical for sources 2 and 3. This heavily contributed to the model predicting unrealistically low or even negative forward temperatures, violating physical bounds. Such violations may propagate through the state iteration procedure and cause numerical failures, for example during mixing temperature calculations.

To correct this, the preprocessing was updated so that all empty entries were replaced with a physically reasonable minimum temperature of 70°C. Additionally, a masked loss function was introduced: for forward temperature variables, the mask was set to 1 only if the corresponding power was non-zero.

Although the masked loss ensured that the network focused on active periods, the POD-based ROM still reconstructs all coefficients, and therefore forward temperatures are always predicted regardless of the source's state. The mask influences the learning target but cannot "turn off" individual reconstructed variables.

Figure 7.10 shows the results obtained after updating the preprocessing.

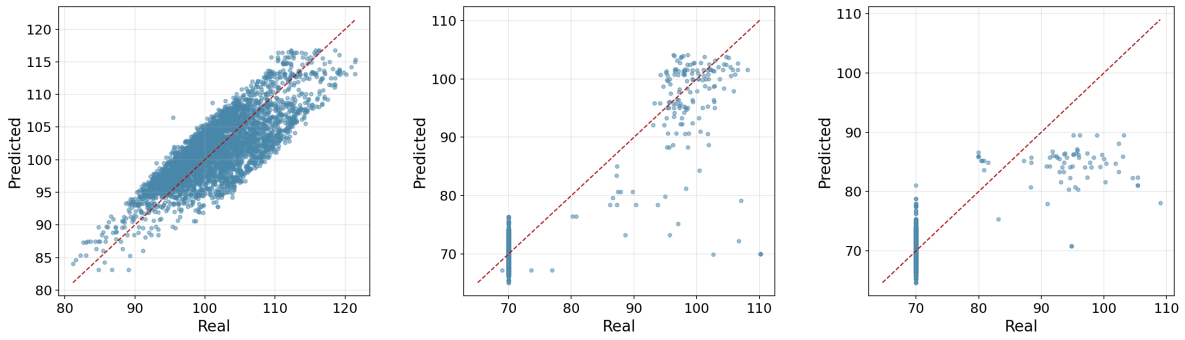


Figure 7.9: Source 1 Source 2 Source 3

Figure 7.10: Predicted versus true values of T_{Fw} for the three sources using preprocessing where NaN values were set to 70°C.

Although the updated preprocessing changes the distribution of predicted temperatures, it does not constitute an improvement. The neural network still fails to recognize when sources 2 and 3 should turn on or off based solely on the temperature data, and instead produces consistently low predictions.

This difficulty is expected: physically, the forward temperatures are not low when a source is inactive, but this information is absent from the FOM training data because these values were irrelevant during off periods. Given that the baseline FOM measurements are missing, the neural network cannot infer the correct behavior.

With the current architecture, POD combined with a neural network predicting temporal coefficients, it is not possible to selectively disable predictions for individual state variables. Therefore, meaningful improvement for this architecture must come from enhancing the training data itself. Potential solutions will be discussed further in section 8.2.

7.6. Conclusion

The objective of this chapter was to evaluate the performance of the ROM on a larger DH network with respect to both accuracy and computational efficiency. The assessment demonstrated that the total mean relative L_2 error defined in (5.4) was 3.62% on the test set, while the corresponding computation time for generating a full prediction was only 0.085 seconds.

Despite the favorable overall error and computation time, the investigation of the forward temperature predictions revealed limitations in the current ROM setup. The analysis showed that the ROM struggled to reproduce physically consistent temperatures for sources that operate only for a limited amount of time, primarily because the FOM training data do not contain valid forward temperature values during off-periods. Consequently, improving the ROM's physical reliability requires enhancements to the underlying training data and model integration, which will be addressed in the next chapter.

8

Conclusion

This thesis investigated whether a ROM can improve the computational efficiency of state variable estimation in DH networks. Although MOR has been applied to DH systems in previous studies, existing approaches either relied on oversimplified representations or lacked scalability, as their applicability depended heavily on the specific DH network topology. As a result, a research gap emerged in designing MOR strategies that combine scalability with computational efficiency, while still delivering accurate state variable estimates for realistic DH networks.

The ROM developed in this work aims to predict the optimal state to which the full network state estimator would converge. Due to the iterative nature of the state estimation process, a non-intrusive approach was adopted. The proposed methodology employs a hybrid POD framework, in which POD compresses the high-dimensional state data into a reduced basis, and a neural network predicts the associated temporal coefficients from the input conditions provided to the network state estimator. To ensure physical consistency, the ROM predictions serve as the initial guess within the state iteration procedure.

This outcome answers the first subquestion by demonstrating how ROM methods can be combined with neural networks in a way that does not require any manual, topology-specific adjustments. POD provides the reduced representation of the state, while the neural network predicts the associated temporal coefficients. The ROM depends on the underlying DH network because both the POD basis and the learned temporal behavior are derived from the FOM data. However, the training pipeline is fully general. Adapting the method to a different DH network only requires providing the corresponding FOM dataset. This method was applied to two test cases to assess its accuracy, computational performance, and resulting speed-up.

In this final chapter, a detailed summary of the main findings for these two test cases is provided and the research questions are revisited in section 8.1. Afterwards, we reflect on the limitations of our approach and highlight directions for future research in section 8.2.

8.1. Summary of findings

Results of the ROM applied to the small test case To answer the second subquestion, which concerns how accurately the ROM predicts the FOM's state variables and how much faster it can compute these states for a realistic DH network, the method was applied to a system with 3 sources, 217 sinks and 1,051 pipes. The ROM was trained on 70% of a 16-day dataset with 15-minute time intervals, selected at random, with the remaining 30% allocated to validation and testing.

Performing POD on the dataset produced the output modes and their corresponding output coefficients. Analysis of the singular value decay revealed a clear elbow at mode 2, where the decay transitions from steep to shallow. Moreover, 99% of the total energy was contained in the first mode, indicating that the dominant variations of the dataset can be captured in a small set of modes.

Determining the optimal number of input and output POD modes is, however, non-trivial, as both

choices influence one another. Increasing the number of output modes reduces the output projection error, yet it also raises the burden on the neural network, which must accurately predict a larger coefficient vector and may therefore introduce higher prediction error. Conversely, adding more input modes provides the neural network with more information, but beyond a certain point the model may struggle to identify the most relevant features. To account for these interactions, a systematic evaluation of multiple input-output mode combinations was performed, tracking the output projection, prediction, and total error. This analysis was conducted over both a broad mode range and a more refined zoom-in region, with four fold cross-validation applied in each case. The optimal configuration was found to consist of 34 input modes and 50 output modes.

In a similar manner, the neural network hyperparameters (learning rate, activation function, and network depth and width) were optimized through a grid search procedure with cross-validation. The resulting architecture and hyperparameter settings are summarized in table 6.5.

The four error metrics (5.2)-(5.4) on the test set for this DH network are provided in table 8.1.

Table 8.1: Mean relative L_2 and relative Frobenius output projection, input projection, prediction and total error for the small DH case on the test set

	Mean rel. error (%)	**Rel. Frobenius error (%)**
Output projection error	$6.12 \cdot 10^{-5}$	$5.94 \cdot 10^{-5}$
Input projection error	0.86	0.83
Prediction error	0.97	0.90
Total error	4.79	0.90

From these results, we conclude that all relative errors remain well below the 5% threshold, demonstrating accuracy performance comparable to that of the FOM.

The full training procedure required approximately four minutes, with roughly half of this time devoted to training the neural network, around 30% to performing POD, and the remainder to data loading and preprocessing. Once trained, the ROM achieved a prediction time of only 0.033 seconds for the entire test set, illustrating a substantial computational advantage while retaining acceptable accuracy. For comparison, a single iteration of the network state estimator can take up to 100 seconds, indicating a significant speed-up.

To answer the third subquestion, whether using the ROM to generate an initial guess can yield a speed-up in optimization runtime, the ROM was integrated into the existing optimization framework. In this configuration, the ROM supplies the initial guess, after which the state iteration converges to a physically consistent solution. The optimization without the ROM required 1401 seconds, whereas the configuration employing the ROM completed in 1202 seconds, corresponding to a speed-up of 1.17. Both approaches yielded nearly identical optimal decision variables; the mean relative L_2 error between them was only 0.02%, indicating that, for this test case, the ROM delivers a meaningful acceleration of the optimization process while preserving a level of accuracy comparable to the FOM.

The optimization procedure consists of an initial candidate (a combination of decision variables) and iterative updates with a decreasing step size, where the update direction is determined by how much "room" remains to lower source temperatures without violating the optimization constraints. For each candidate (up to a predetermined maximum) the network state estimator is executed and a candidate is considered converged only when all time steps satisfy the convergence criteria.

For this test case, two convergence criteria are evaluated: the deviation in total sink power and the deviation in total shunt flow, each required to fall below a predefined tolerance relative to the total system load. An analysis of the convergence behavior showed that, after approximately the first five candidates, the ROM configuration consistently reduced the number of iterations needed to achieve convergence by 1, 2, or even 3 iterations. Moreover, evaluating the distribution of the convergence criteria across time steps revealed that, for later candidates, fewer time steps required updating.

Finally, in the ROM-based configuration, the initial guess for each time step is computed directly by the ROM, whereas in the FOM-based configuration an additional iteration of the state estimator is required to generate this initial state. In effect, the ROM "skips" this first iteration by providing an

inexpensive prediction. Together, these three effects explain the observed speed-up achieved by the ROM configuration.

For this test case we can therefore conclude that the ROM served as an effective surrogate model that accelerated the optimization procedure without compromising solution quality.

Results of the ROM applied to the large test case The larger test case also includes three sources but features 427 additional pipes and nearly three times as many sinks as the previous network. The corresponding dataset spans a 28-day period with 15-minute time intervals and was again divided into three randomly assigned subsets: 70% for training, 15% for validation, and 15% for testing. Hyperparameter tuning and POD mode optimization were repeated for this system, resulting in an optimal configuration of 40 input modes and 42 output modes, with all remaining optimized hyperparameters and settings reported in table 7.5.

The four intermediate error metrics (5.2)-(5.4) associated with the POD reconstruction and neural-network prediction for this test case are presented in table 8.2.

Table 8.2: Mean relative L_2 and relative Frobenius output projection, input projection, prediction and total error for the large DH case on the test set

	Mean rel. error (%)	**Rel. Frobenius error (%)**
Output projection error	$1.81 \cdot 10^{-5}$	$1.82 \cdot 10^{-5}$
Input projection error	1.20	1.20
Prediction error	0.99	1.88
Total error	3.62	1.88

Again, we conclude that all relative errors remain well below the 5% threshold, demonstrating accuracy performance comparable to that of the FOM. The training time increased to slightly over 20 minutes, which is expected given that the dataset contains more than two and a half times as many samples and the network architecture is almost three times larger. Nevertheless, this training effort can be seamlessly integrated into the existing workflow, which can take up to 12 hours, meaning the additional time is not expected to pose any practical difficulties. More importantly, the prediction time for the test set was only 0.085 seconds.

However, when integrating the ROM into the optimization framework for this test case, several issues emerged. When analyzing the resulting decision variables, it became evident that no meaningful optimal values were obtained for the third source. A closer inspection of the neural network predictions after reconstruction revealed that the model sometimes generated negative forward temperatures for sources 2 and 3. This behavior could be traced back to the preprocessing step in which missing temperature values in the training set were automatically replaced with zeros.

Further analysis of the training data showed that sources 2 and 3 functioned only as back-up units and were active for a very small fraction of the dataset, approximately 4% of all time steps for source 2 and only 2% for source 3. Consequently, the neural network was predominantly exposed to situations in which these sources were effectively “off,” and the temperature entries were zero. As a result, the network learned that extremely low temperatures represented the default operational state, leading to physically unrealistic forward temperature predictions and, ultimately, incorrect optimization results.

Replacing the zero-value preprocessing with a more realistic minimum temperature of 70°C, and introducing a masked loss function to distinguish between on- and off-states, did not yield a improvement. The neural network continued to underestimate the forward temperatures of the intermittently active sources.

Because the ROM is built upon a POD decomposition, in which the system dynamics are represented through a fixed set of spatial modes and temporal coefficients, the ROM must always reconstruct a complete physical state, regardless of whether individual components of the system are active or inactive. As a consequence, the neural network is required to output temporal coefficients corresponding to all state variables at every time step, including those for which no physically meaningful value exists in the training data. As a result, the neural network has no opportunity to learn the actual thermal behavior of back-up sources during off-periods, nor can it infer when the sources are turned on.

Therefore, the effectiveness of the ROM as a surrogate in this test case is influenced by the extent to which the training data represent the different operating states of the sources.

Taken together, the results of both test cases allow us to answer the main research question. A ROM can enhance the computational efficiency of state variable estimation in DH networks, provided that the underlying training data sufficiently represent the relevant operating regimes of the system.

8.2. Discussion and future work

The difficulties encountered in predicting forward temperatures for the large test case reveal an important methodological limitation in the current ROM approach. Although the neural network and the POD-based ROM can capture the dominant behavior of the DH network variables, they rely fundamentally on the quality and physical consistency of the training data. In this test case, the FOM dataset did not contain valid forward temperature values for back-up sources when they were switched off; instead, these entries were stored as missing values and later replaced during preprocessing with constant values. As a result, the ROM was trained on data that was not fully physically consistent.

The POD reconstruction forces the ROM to output a full temperature field at every time step, regardless of whether a source is active. This architectural constraint prevents the model from selectively ignoring variables whose physical meaning is undefined in the snapshots. Consequently, the ROM consistently underestimated the forward temperatures of back-up sources in the large test case.

The small test case also contained sources that were inactive for parts of the time horizon. However, unlike in the large test case, these sources were active for more than 50% of the data, giving the ROM sufficient exposure to their on-state behavior. This suggests that the ROM may indeed be able to learn the forward temperatures accurately when the training dataset contains enough representative samples of active periods.

Even though insufficient activation data is the most plausible explanation for the degraded performance, this has not yet been conclusively verified. A next step would be to train the ROM on curated datasets where all three back-up sources are active for a significant fraction of the time steps and then examine the resulting errors and the optimization behavior. By repeating this for several controlled datasets with, for example, 10%, 20%, and 50% activation time, one can determine the minimum percentage of ”turned-on” data the neural network requires to learn the activation behavior reliably. Other edge cases should also be explored, such as extreme peaks in demand or unexpected weather transitions since these may expose additional weaknesses that are not captured in the current experiments.

Another direction for future work is to explicitly provide operational information of the sources as an additional input to the neural network. In the current setup, the model must infer on/off behavior solely from temperature and power data. Introducing a binary on/off indicator per source would allow the network to condition its predictions on the operational state, potentially enabling it to ignore temperature dynamics for inactive sources.

A second improvement could involve replacing the feedforward neural network with a self-attention neural network architecture. Self-attention neural networks dynamically weight the relevance of different input features and could therefore be better suited for datasets where certain variables are meaningful only under specific operating conditions [65]. A self-attention model could suppress the influence of non-physical temperature values during off-states and better capture the nonlinear switching behavior of back-up sources.

The limitations described above introduce the need for an initial quality check of the training data before the ROM can be used in an optimization setting. A practical way to formalize this check is to evaluate the source forward temperature prediction error, since the source temperatures act directly as decision variables. A threshold can be implemented using a relative error metric (e.g., the relative error of T_{Fw} for the sources must remain below 10%) and if this value exceeds a chosen limit, the ROM should not be used or the training data must be extended. This diagnostic can naturally be integrated into a multi-fidelity framework such as in [33], where the ROM is employed only when the diagnostic error is sufficiently low and the FOM is used otherwise.

Another limitation of the current framework is that the neural network is trained on a subset of the

snapshot matrix rather than on the actual inputs used during the state iteration. As a consequence, the network may partially fall back on identity mappings for variables that, in the full iterative solver, are themselves updated at every iteration. A potential better training strategy would therefore require exposing the network to the true inputs encountered during the state iteration. These inputs are not contained in the historical optimization dataset, but they can be reconstructed by running the network state estimator and recording, at each iteration, both the inputs it receives and the outputs it produces. This procedure would generate a new dataset that directly reflects the quantities passed to the iterative solver.

Collecting such data, however, entails executing the FOM repeatedly until a sufficiently rich set of state iteration inputs has been obtained, which may be computationally expensive. The feasibility of this approach depends on how much data is actually required to capture the relevant system behavior. If the system behavior of interest (e.g. edge cases, on/off behavior of sources) is known in advance, the required dataset can be assembled from a smaller set of representative operating points. In that case, the additional cost of generating dedicated training data may be justified, particularly if it leads to more accurate ROM predictions enabling a substantial speed-up by reducing the amount of iterations even further.

The current ROMs were trained on the available consecutive datasets, usually spanning several days or a full month. For the small test case, the optimization scenario occurred within the same month (December) as the training data. This means that the seasonal characteristics of the training and evaluation periods were very similar, with comparable temperature profiles and heating demands. Future studies should therefore include experiments across multiple seasons to ensure that the ROM generalizes beyond the specific conditions it was trained on. Furthermore, although the dataset for the large network spanned an entire month, it is possible that much of this data is redundant. As DH networks often follow highly repetitive daily profiles, a large dataset may exhibit low information diversity. One way to address this is to construct a reduced training set using a greedy selection strategy: starting from a small subset, the model is repeatedly evaluated on the remaining data, and the sample with the highest prediction error is added to the training set. This approach prioritizes the most informative samples, potentially reducing training time without sacrificing accuracy.

An interesting result of the mode optimization is that the lowest prediction error occurs with a configuration of approximately 30 to 50 input and output POD modes, far above the elbow point in the singular value decay (mode 2 for the small case and mode 3 for the large case). While singular value decay and energy capture criteria indicate how well the modes reconstruct the data, they do not reveal how predictable each mode is from the system's inputs. Low energy modes can still encode spatial structures that are highly sensitive to input changes. The inherent global nature of POD can, therefore, require many nodes to represent localized dynamics. A promising direction for future work is to investigate alternatives to the POD basis itself. Methods such as greedy reduced basis techniques [47] or kernel based POD variants [55] may offer more compact or more predictive representations, and exploring their implementation within the current ROM architecture could lead to a smaller reduced basis.

While the ROM achieved a computational speed-up for the small test case, the optimizer still required several iterations per candidate to reach convergence. This behavior is likely linked to the fact that the ROM does not enforce physical constraints such as local and global mass conservation or nonlinear pressure-flow relations. As a result, the optimizer must correct deviations from physical feasibility in successive iterations. For the later candidates of the small test case, convergence was reached in only two to three iterations, and achieving this consistently would represent a meaningful improvement over the FOM.

A valuable extension of this work would therefore be to analyze the degree to which individual physical constraints are violated during each iteration. Such an investigation could reveal which variables or components are most poorly represented by the ROM and where the dominant physical inconsistencies originate. Furthermore, understanding the relationship between prediction accuracy and convergence behavior could give more insights: errors in some quantities may have only minor effects, whereas inaccuracies in others may trigger large feasibility corrections and thus prolong the iterative process. This relationship could be studied by tracking the prediction errors per (type of) variable alongside the corresponding constraint violations and iteration counts. Identifying which predictions most strongly influence convergence would make it possible to target model improvements where they yield the

greatest reduction in iteration count, and applying this analysis to the large test case in particular may offer further insight into how accuracy limitations and constraint violations influence the convergence.

Incorporating physics-informed constraints, such as enforcing conservation laws, could reduce the correction effort and improve convergence. Many of these constraints can be expressed as a system of linear equations. This structure makes it possible to enforce physical consistency by applying a suitable linear transformation to the ROM predictions. Such a transformation can project any unconstrained output onto the subspace that satisfies the linear conservation laws, reducing the optimizer's need to correct physically inconsistent states in subsequent iterations. However, implementing such constraints would introduce topology dependent components into the ROM, reducing the uniformity and simplicity of the current training pipeline.

Overall, the evaluation shows that the reliability of the ROM depends strongly on the representativeness and completeness of the training snapshots. Future research should focus on assessing and enhancing the physical consistency of the ROM, identifying or designing more effective reduced bases, and determining the most reliable and efficient composition of the training dataset.

References

[1] Rodrigo Abadía-Heredia et al. “A predictive hybrid reduced order model based on proper orthogonal decomposition combined with deep learning architectures”. In: *Elsevier* 187 (2022).

[2] Charu C. Aggarwal. *Neural Networks and Deep Learning*. Springer, 2023.

[3] Aaron D Ames, Joe Moeller, and Paulo Tabuada. “Categorical Lyapunov Theory I: Stability of Flows”. In: *arXiv* (2025).

[4] Zeynab Azarhoosh and Majid Ilchi Ghazaan. “A Review of Recent Advances in Surrogate Models for Uncertainty Quantification of High-Dimensional Engineering Applications”. In: *Computer Methods in Applied Mechanics and Engineering* (2024).

[5] Taha Boussaid et al. “Enabling fast prediction of district heating networks transients via a physics-guided graph neural network”. In: *Applied Energy* (2024).

[6] Melissa C. Brindise and Pavlos P. Vlachos. “Proper orthogonal decomposition truncation method for data denoising and order reduction”. In: *Experiments in fluids* (2017).

[7] Glenn O. Brown. “The History of the Darcy-Weisbach Equation for Pipe Flow Resistance”. In: *ASCE* (2004).

[8] Sandeep Reddy Bukka et al. “Assessment of unsteady flow predictions using hybrid deep learning based reduced order models”. In: *Physics of Fluids* (2021).

[9] Djork-Arne Clevert, Thomas Unterthiner, and Sepp Hochreiter. “Fast and accurate deep network learning by exponential linear units (ELUs)”. In: *arXiv* (2015).

[10] Adrián Corrochano et al. “A Predictive Physics-Aware Hybrid Reduced Order Model for Reacting Flows”. In: *Numerical Methods in Engineering* (2025).

[11] L. Minh Dang et al. “Toward explainable heat load patterns prediction for district heating”. In: *Scientific Reports* (2023).

[12] Matthias Eimer. “Modeling and Efficient Simulation of District Heating Networks”. PhD thesis. Technische Universität Kaiserslautern, 2021, p. 15.

[13] European Commission. *Commission Staff Working Document: Impact Assessment Accompanying the Document Proposal for a Regulation of the European Parliament and of the Council Establishing the Framework for Achieving Climate Neutrality and Amending Regulation (EU) 2018/1999 (European Climate Law)*. 2020. URL: `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52020SC0176` (visited on 07/21/2025).

[14] European Environment Agency. *Decarbonising heating and cooling — a climate imperative*. 2023. URL: `https://www.eea.europa.eu/publications/decarbonisation-heating-and-cooling` (visited on 07/21/2025).

[15] Basak Falay, Ingo Leusbrock, and Carles Ribas Tugore. “Improving the computational performance of district heating network simulation”. In: *International Journal of Sustainable Energy* (2024).

[16] Antonella Falini. “A review on the selection criteria for the truncated SVD in Data Science applications”. In: *Journal of Computational Mathematics and Data Science* (2022).

[17] M. Frangos et al. “Surrogate and reduced-order modeling: a comparison of approaches for large-scale statistical inverse problems”. In: *Computational Methods for Large-Scale Inverse Problems and Quantification of Uncertainty* (2010).

[18] David Furtado. “Modelling of a District Heating System with Decentralized Heat Generators”. MA thesis. Technische Universität München, 2015.

[19] Laura Boca de Giuli, Alessio La Bella, and Riccardo Scattolini. “Physics-informed Neural Network Modelling and Predictive Control of District Heating Systems”. In: *IEEE Transactions on Control Systems Technology* 32 (2024).

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[21] Gradyent. *Upgrade Your Heating Grid with Our Powerful Digital Twin*. 2025. URL: https://www.gradyent.ai/ (visited on 07/21/2025).

[22] Elisa Guelpa et al. “Optimal operation of large district heating networks through fast fluid-dynamic simulation”. In: *Energy* 102 (2016).

[23] Elisa Guelpa et al. “Thermal load prediction in district heating systems”. In: *Energy* (2019).

[24] Alaeddine Hajri et al. “Data-driven model for heat load prediction in buildings connected to district heating networks”. In: *Energy* (2025).

[25] Dan Hendrycks and Kevin Gimpel. “Gaussian Error Linear Units (GELUs)”. In: *arXiv* (2023).

[26] Jan S. Hesthaven and Stefano Ubbiali. “Non-intrusive reduced order modeling of nonlinear problems using neural networks”. In: *Journal of Computational Physics* (2018).

[27] HVAC. *Understanding District Heating: Efficient Energy Distribution Systems for Sustainable Urban Applications – Technical Overview, Installation, Maintenance, and Standards for HVAC-R Professionals*. URL: https://hvac-eng.com/understanding-district-heating-efficient-energy-distribution-systems-for-sustainable-urban-applications-technical-overview-installation-maintenance-and-standards-for-hvac-r-professionals/ (visited on 12/03/2025).

[28] Mengting Jiang et al. “A data-based reduced-order model for dynamic simulation and control of district-heating networks”. In: *Applied Energy* 340 (2023).

[29] Victor Richmond R. Jose. “Percentage and Relative Error Measures in Forecast Evaluation”. In: *Informs* (2017).

[30] Ali Kafalı et al. *Overview of surrogate modelling approaches, including an initial risk analysis on standards for re-usability and transferability*. Tech. rep. SmartEM Project, Philips Consumer Lifestyle B.V., 2025.

[31] Malick Kane and Jérémy Rolle. ““Quantum networks”: a new approach for representing a network and evaluating hydraulic and thermal losses in district heating/cooling systems”. In: *Proceedings of ECOS 2020 - The 33rd International Conference on Efficiency, Cost, Optimization, Simulation and Environmental Impact of Energy Systems*. 2020.

[32] Winfried Keiper, Anja Milde, and Stefan Volkwein. *Reduced-Order Modeling (ROM) for Simulation and Optimization*. Springer, 2018.

[33] Benedikt Klein and Mario Ohlberger. “Multi-fidelity Learning of Reduced Order Models for Parabolic PDE Constrained Optimization”. In: *arXiv* (2025).

[34] Tim Krake, Daniel Weiskopf, and Bernhard Eberhardt. “Dynamic Mode Decomposition: Theory and Data Reconstruction”. In: *CoRR* (2022).

[35] Saltanat Kuntuarova et al. “Design and simulation of district heating networks: A review of modeling approaches and tools”. In: *Energy* (2024).

[36] Helge V Larsen, Benny Bøhm, and Michael Wigbels. “A comparison of aggregated models for simulation and operational optimisation of district heating networks”. In: *Energy Conversion and Management* (2004).

[37] Helge V. Larsen et al. “Aggregated dynamic simulation model of district heating networks”. In: *Energy Conversion and Management* (2002).

[38] Dominik Linn. “Modeling and Analysis of District Heating Networks with Applications in Optimal Control”. PhD thesis. Technische Universität Kaiserslautern, 2022, p. 14.

[39] Bo Liu and Zhengchun Du. “Krylov subspace method for passive reduced-order modelling of electrical networks”. In: *IET Generation, Transmission Distribution* (2017).

[40] Jiaqi Luo. “Design optimization of the last stage of a 4.5-stage compressor using a POD-based hybrid model”. In: *Aerospace Science and Technology* (2018).

[41] Muhammad Haris Malik. “Reduced Order Modeling for Smart Grids’ Simulation and Optimization”. PhD thesis. Institut de Calcul Intensif (ICI), 2017.

[42] Jan Mohring et al. “Optimal control of district heating networks using a reduced order model”. In: *Optimal Control Applications and Methods* 41 (2020), pp. 1352–1370.

[43] William L. Oberkampf and Christopher J. Roy. *Verification and Validation in Scientific Computing*. Cambridge University Press, 2010.

[44] Mario Ohlberger and Stephan Rave. “Reduced Basis Methods: Success, Limitations and Future Challenges”. In: *Proceedings of Algoritmy* (2016), pp. 1–12.

[45] Guglielmo Padula, Michele Girfoglio, and Gianluigi Rozza. “A brief review of Reduced Order Models using intrusive and non-intrusive techniques”. In: *Proceedings in Applied Mathematics and Mechanics* 24 (2024).

[46] Planète Énergies. *District Heating Systems: Uneven Use Around the World*. 2015. URL: `https://www.planete-energies.com/en/media/article/district-heating-systems-uneven-use-around-world` (visited on 07/21/2025).

[47] Alfio Quarteroni, Andrea Manzoni, and Federico Negri. *Reduced Basis Methods for Partial Differential Equations*. Springer, 2016.

[48] Alfio Quarteroni and Gianluigi Rozza. *Reduced Order Methods for Modeling and Computational Reduction*. Springer, 2014.

[49] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* (2019).

[50] Wajdi Rajhi et al. “Use of proper orthogonal decomposition and machine learning for efficient blood flow prediction in cerebral saccular aneurysms”. In: *Scientific Reports* (2025).

[51] Markus Rein. “Order Reduction for Nonlinear Dynamic Models of District Heating Networks”. PhD thesis. Technische Universität Kaiserslautern, 2019.

[52] Thibaut Résimont et al. “Economic and environmental comparison of a centralized and a decentralized heating production for a district heating network implementation”. In: *Conference: 10th International Conference on System Simulation in Buildings*. 2018.

[53] Dubon Rodrigue et al. “Topology reduction through machine learning to accelerate dynamic simulation of district heating”. In: *Energy and AI* (2024).

[54] Carlos J.G. Rojas, Andreas Dengel, and Mateus Dias Ribeiro. “Reduced-order Model for Fluid Flows via Neural Ordinary Differential Equations”. In: *arXiv* (2021).

[55] Matteo Salvador, Luca Dedè, and Andrea Manzoni. “Non intrusive reduced order modeling of parametrized PDEs by kernel POD and neural networks”. In: *Computers Mathematics with Applications* (2021).

[56] Wilhelmus H.A. Schilders, Henk A. van der Vorst, and Joost Rommes. *Model Order Reduction*. Springer, 2008.

[57] Philipp Schulze. “Structure-Preserving Model Reduction for Port-Hamiltonian Systems Based on a Special Class of Nonlinear Approximation Ansatzes”. In: *arXiv* (2023).

[58] Johan Simonsson, Khalid Tourkey Atta, and Wolfgang Birk. “Reduced-Order Modeling of Thermal Dynamics in District Energy Networks using Spectral Clustering”. In: *IEEE* (2022).

[59] Renee Swischuka et al. “Projection-based model reduction: Formulations for physics-based machine learning”. In: *Computer and Fluids* (2019).

[60] Jonathan H. Tu, Clarence W. Rowly, and Dirk M. Luchtenberg. “On Dynamic Mode Decomposition: Theory and Applications”. In: *Journal of Computational Dynamics* 1 (2014), pp. 391–421.

[61] Peter Wallentén. “Steady-state heat loss from insulated pipes”. PhD thesis. Lund University, 1991.

[62] Deming Wang. “Heat loss along the pipeline and its control measures”. In: *SN Applied Sciences* 5 (2022).

[63] Qian Wang, Jan S. Hesthaven, and Deep Ray. “Non-intrusive reduced order modeling of unsteady flows using artificial neural networks with application to a combustion problem”. In: *Journal of Computational Physics* (2019).

[64] Daniel N. Wilke. “Multifidelity Surrogate Models: A New Data Fusion Perspective”. In: *arXiv* (2024).

[65] Weile Xu et al. “Real-time rocket motor internal flow field prediction based on hybrid POD and deep neural networks”. In: *International Journal of Heat and Mass Transfer* (2026).

[66] Yu Zhang et al. “Reduced-order modelling of unsteady cavitating flow around a Clark-Y hydrofoil”. In: *Physics of Fluids* (2025).

A

Overview of referenced ROM methods and error metrics

A.1. Dynamic Mode Decomposition (DMD)

The Dynamic Mode Decomposition (DMD) method begins by collecting a sequence of snapshots at successive time steps. Given a set of data vectors $\{x_0, x_1, \ldots, x_m\}$, where each $x_i \in \mathbb{R}^n$, these snapshots are arranged into two matrices:

$$X = \begin{bmatrix} | & | & & | \\ x_0 & x_1 & \cdots & x_{m-1} \\ | & | & & | \end{bmatrix}, \quad Y = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \cdots & x_m \\ | & | & & | \end{bmatrix}, \quad X, Y \in \mathbb{C}^{n \times m}.$$

The objective of DMD is to identify a linear operator A such that $Y \approx AX$ [34]. However, in systems with complex dynamics the state dimension n can be very large, often in the order of thousands. This would make the direct computation of $A \in \mathbb{C}^{n \times n}$ computationally prohibitive.

Therefore DMD uses the SVD (see section 3.3.2) of X

$$X = U \Sigma V^*,$$

where $U \in \mathbb{C}^{n \times r}$, $\Sigma \in \mathbb{C}^{r \times r}$, and $V \in \mathbb{C}^{m \times r}$, with r being the rank of X. The reduced operator $\tilde{A} \in \mathbb{C}^{r \times r}$ is then defined as

$$\tilde{A} = U^* Y V \Sigma^{-1}.$$

Solving the eigenvalue problem

$$\tilde{A} w = \lambda w$$

yields the eigenvalues λ and eigenvectors w of the reduced system. The corresponding DMD modes are computed as

$$\Phi = Y V \Sigma^{-1} W,$$

where W contains the eigenvectors of $\tilde{A}$ [60]. This approach avoids the explicit computation of the high-dimensional matrix A, significantly reducing computational cost. The resulting DMD modes capture the dominant spatial-temporal modes of the system and can form the reduced basis of a ROM.

A.2. Moment matching

Moment matching is widely used for MOR of large-scale linear time-invariant (LTI) systems, especially in control engineering and electronic circuits. The core idea is to construct a ROM whose transfer function matches a prescribed number of moments of the original system's transfer function around one or more expansion points in the frequency domain [48].

Consider the generalized LTI system in the first-order form

$$\begin{aligned} E\dot{x}(t) &= Ax(t) + Bu(t), \\ y(t) &= Cx(t) \end{aligned}$$

where $E, A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, $x(t) \in \mathbb{R}^n$ is the state, $u(t) \in \mathbb{R}^m$ is the input and $y(t) \in \mathbb{R}^p$ is the output [56]. Such formulations, for example, naturally arise in linearized electric circuits.

Taking the Laplace transform with zero initial conditions yields the *transfer function*

$$G(s) = C(sE - A)^{-1}B. \tag{A.1}$$

This function describes the input-output behavior of the system in the frequency domain.

The goal of moment matching is thus to construct a ROM whose transfer function $G_r(s)$ matches the first several derivatives (moments) of $G(s)$ at one or more selected expansion points.

Let $s_0 \in \mathbb{C}$ be an expansion point and assume that $A - s_0E$ is nonsingular. Expanding $G(s)$ into a Taylor series around s_0 gives

$$\begin{aligned} G(s) &= -C(I - (s - s_0)(A - s_0E)^{-1}E)^{-1}(A - s_0E)^{-1}B \\ &= M_0 + M_1(s - s_0) + M_2(s - s_0)^2 + \dots \end{aligned}$$

where the matrices

$$M_j = -C((A - s_0E)^{-1}E)^j(A - s_0E)^{-1}B, \quad \text{for } j = 0, 1, 2, \dots$$

are called the j *moments* of the system at the point s_0. These moments correspond to the derivatives $M_j = \frac{1}{j!}\frac{d^j}{ds^j}G(s)\Big|_{s=s_0}$.

The reduced system of dimension $r << n$ is defined as

$$\begin{aligned} E_r\dot{x}_r(t) &= A_rx_r(t) + B_ru(t), \\ y_r(t) &= C_rx_r(t) \end{aligned}$$

with transfer function

$$G_r(s) = C_r(sE_r - A_r)^{-1}B_r$$

The ROM is said to match the first q moments of the full system at s_0 if

$$M_j = M_{r,j}, \quad \text{for } j = 0, \dots, q$$

where $M_{r,j}$ are the Taylor coefficients of $G_r(s)$.

Moment matching yields a ROM that accurately reproduces the system's low- and mid-frequency dynamics and approximates the impulse and transient responses well. Moreover, when the FOM is stable and passive, these properties are preserved in the ROM as well [56].

A.3. Petrov-Galerkin projection

The Petrov-Galerkin method operates similarly to the Galerkin method described in section 3.3.3, with one key distinction: it uses a test space different from the trial space [47]. Although an approximate solution is still sought in the reduced trial space $V_r \subset V$, the residual is now enforced to be orthogonal to a separate test space $W_j \subset W$, where W is also a Hilbert space and W_j is a finite-dimensional subspace spanned by the basis functions $\{\psi_1, ..., \psi_j\}$.

Hence, enforcing the Petrov-Galerkin condition yields

$$a(x_r(t), w) = f(w) \quad \forall w \in W_r.$$

Substituting $x_r(t) = \sum_{j=1}^{r} \alpha_j(t)\phi_j$ into the bilinear form gives

$$a\left(\sum_{j=1}^{r} \alpha_j \phi_j, \psi_i\right) = \sum_{j=1}^{r} \alpha_j a(\phi_j, \psi_i) = f(\psi_i).$$

This leads to the system

$$A\tilde{\alpha} = \tilde{b}$$

where $A \in \mathbb{R}^{l \times r}$ with entries $A_{ij} = a(\phi_j, \psi_i)$ and $\tilde{b} = [f(\psi_1), \ldots, f(\psi_r)]^T$. Solving this system for $\tilde{\alpha}$ yields the temporal coefficients.

The Galerkin method is well-suited for symmetric and well-posed problems. In Petrov-Galerkin, flexibility is gained by choosing W_j differently, which can improve stability and accuracy in problems for example containing advection-dominated PDEs [47].

A.4. Kernel POD (KPOD)

Kernel POD (KPOD) is a nonlinear extension of POD that performs dimensionality reduction by implicitly mapping the snapshots to a high-dimensional feature space. In this space, the solution manifold becomes more linearly separable, allowing a low-dimensional linear basis to approximate the nonlinear behavior more efficiently than standard POD [55]. Importantly, this feature space is never constructed explicitly: by applying the kernel trick, all computations rely only on kernel evaluations between snapshots.

Let

$$X = [x(t_1), x(t_2), ..., x(t_m)] \in \mathbb{R}^{n \times m}$$

be the snapshot matrix, and denote its columns by $s_i = x(t_i)$. KPOD replaces the inner products $s_i^T s_j$ used in POD by a kernel function

$$K_{i,j} = \kappa(s_i, s_j)$$

where $\kappa : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a symmetric positive semidefinite kernel. This yields the kernel matrix

$$K = [K_{ij}] \in \mathbb{R}^{m \times m}.$$

Similarly to POD, KPOD performs eigendecomposition of the kernel matrix

$$Kw_k = \lambda_k w_k, \quad \text{for } k = 1, \ldots, m$$

where the eigenvalues are ordered as

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m \geq 0,$$

and we define $\sigma_k = \sqrt{\lambda_k}$.

Because the feature space is implicit and high-dimensional, KPOD obtains representations of the feature-space principal components expressed back in the original snapshot space. For each eigenpair (λ_k, w_k),

$$p_k = \frac{1}{\sigma_k} X w_k \quad \text{for } k = 1, \ldots, m$$

is defined. Collecting these vectors gives

$$P = [p_1, \ldots, p_m] \in \mathbb{R}^{n \times m}.$$

Unlike in POD, the columns of P are not orthonormal. Therefore, a QR factorization is performed to obtain an orthonormal basis $\mathbb{Q}$. The KPOD reduced basis is obtained by selecting the first r columns

of $\mathbb{Q}$. The value of r is chosen through the same relative information criterion (see section 3.3.1) used for POD, replacing POD singular values by the kernel singular values σ_k.

Compared to POD, KPOD can represent nonlinear snapshot manifolds more effectively because the nonlinear mapping increases linear separability. As a result, KPOD may require significantly fewer modes than POD to reach the same accuracy. Another advantage is that KPOD never constructs the high-dimensional feature space explicitly: the method computes only the projections of the snapshots onto the principal components, not the principal components themselves [55].

A.5. Error metrics

- *Root Mean Squared Error (RMSE)*: This metric provides a measure of the average magnitude of the error, expressed in the same units as the original data. It is defined as

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|x_{\text{FOM}}^{(i)} - x_{\text{ROM}}^{(i)}\|_2^2}, \tag{A.2}$$

  where $\|\cdot\|_2$ denotes the L_2 norm and N is the number of data points. RMSE is particularly useful when emphasizing larger errors.

- *Residual error*: This measures how well the ROM satisfies the governing equations of the original system. It is particularly relevant in Galerkin-type methods and PDE-based models and is defined as

$$\text{residual} = \frac{1}{N}\sum_{i=1}^{N} \|f(x_{\text{ROM}}^{(i)}, t_i, \mu_i)\|_2,$$

  where $\|\cdot\|_2$ denotes L_2 norm, $f(\cdot)$ represents the residual of the governing equations and t_i and μ_i refer to the time and parameter sample at index i, respectively. For a linear system, this reduces to

$$\text{residual} = \frac{1}{N}\sum_{i=1}^{N} \|Ax_{\text{ROM}}^{(i)} - b^{(i)}\|_2,$$

  where A represents the system matrix and $b^{(i)}$ represents the forcing vector from the FOM.

B

Convergence ratio distribution plots for all candidates of the small test case

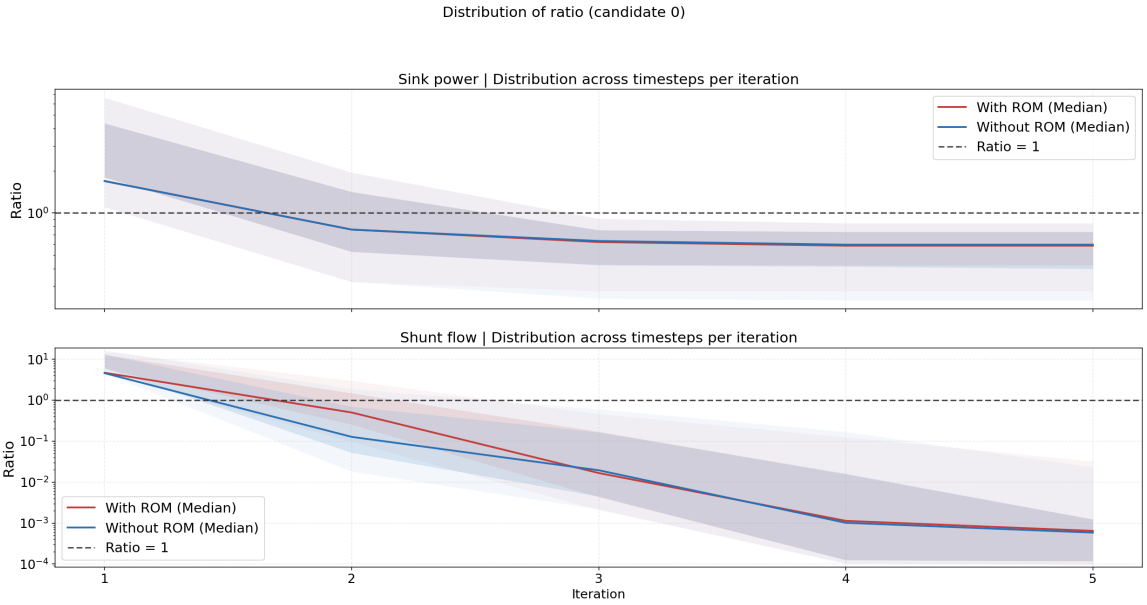


Figure B.1: Distribution of the convergence ratio for the sink power (top) and shunt flow (bottom) criterion across all time steps per iteration for candidate 0. Solid lines show the median ratio per iteration for the simulations with and without the ROM, while the shaded regions indicate the spread across time steps.

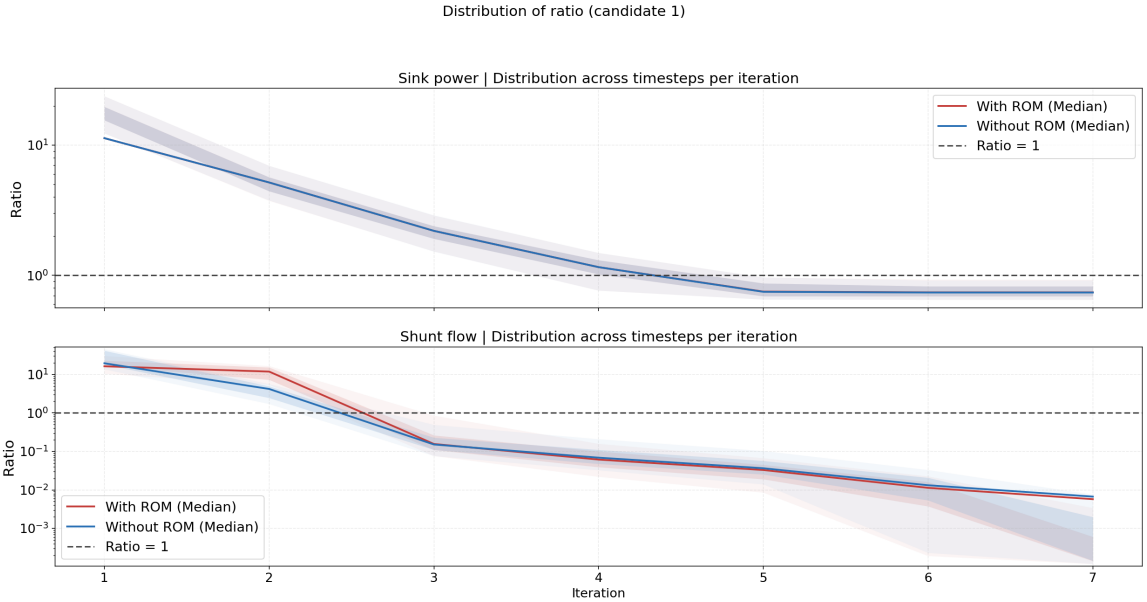


Figure B.2: Distribution of the convergence ratio for the sink power (top) and shunt flow (bottom) criterion across all time steps per iteration for candidate 1. Solid lines show the median ratio per iteration for the simulations with and without the ROM, while the shaded regions indicate the spread across time steps.

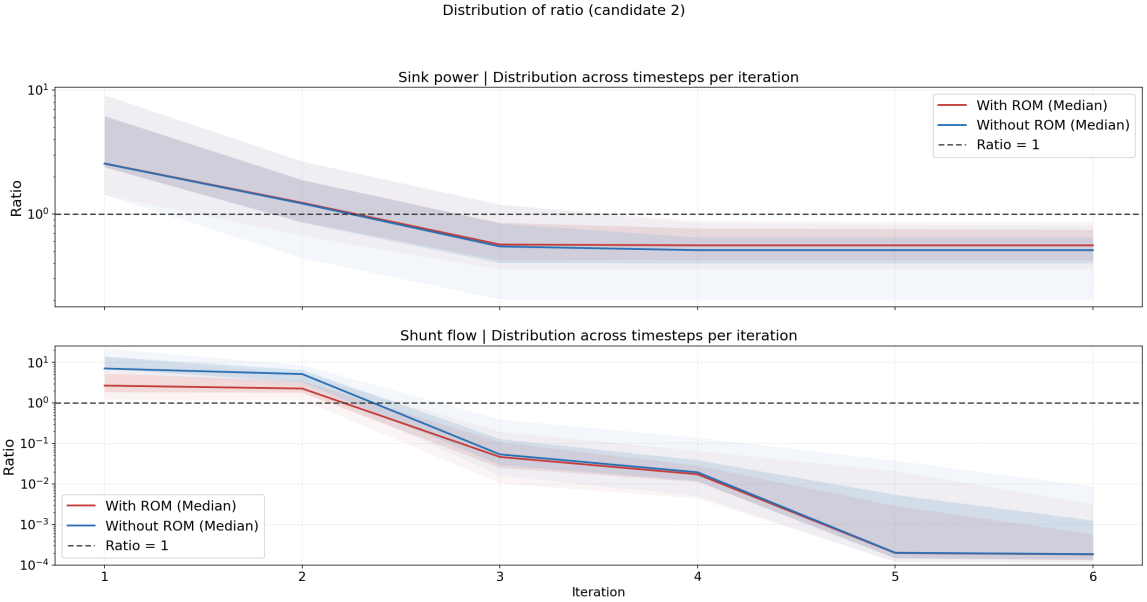


Figure B.3: Distribution of the convergence ratio for the sink power (top) and shunt flow (bottom) criterion across all time steps per iteration for candidate 2. Solid lines show the median ratio per iteration for the simulations with and without the ROM, while the shaded regions indicate the spread across time steps.

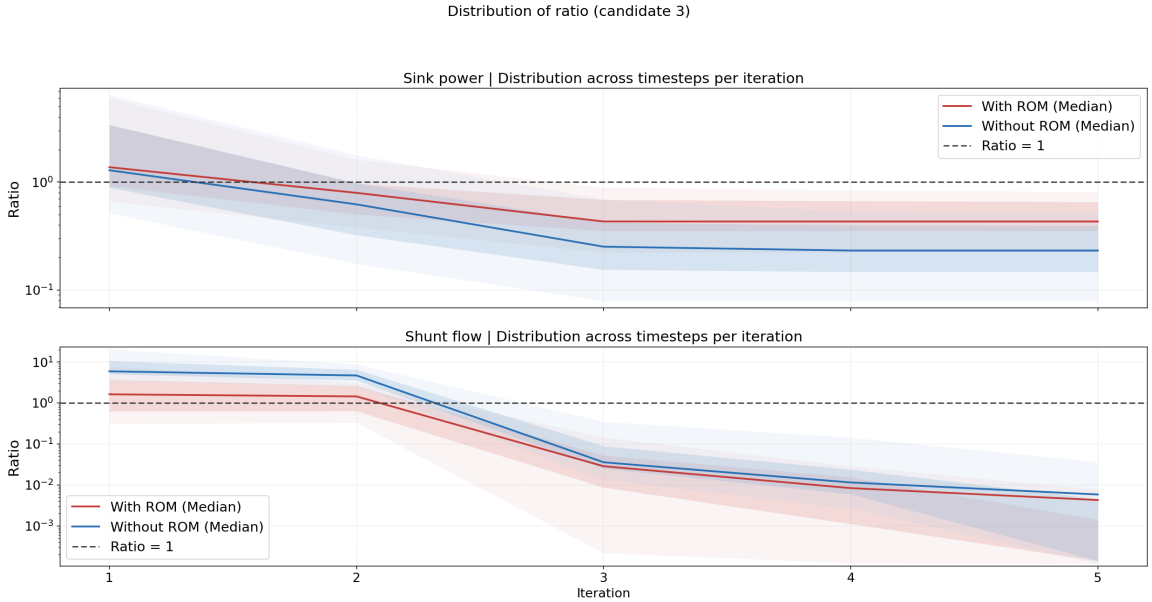


Figure B.4: Distribution of the convergence ratio for the sink power (top) and shunt flow (bottom) criterion across all time steps per iteration for candidate 3. Solid lines show the median ratio per iteration for the simulations with and without the ROM, while the shaded regions indicate the spread across time steps.

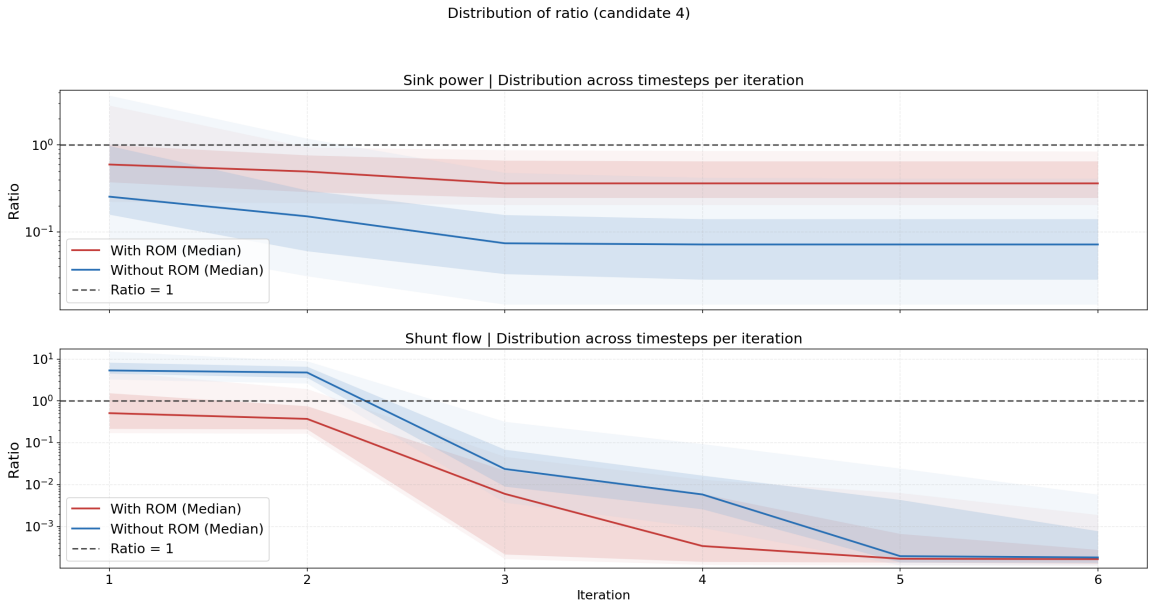


Figure B.5: Distribution of the convergence ratio for the sink power (top) and shunt flow (bottom) criterion across all time steps per iteration for candidate 4. Solid lines show the median ratio per iteration for the simulations with and without the ROM, while the shaded regions indicate the spread across time steps.

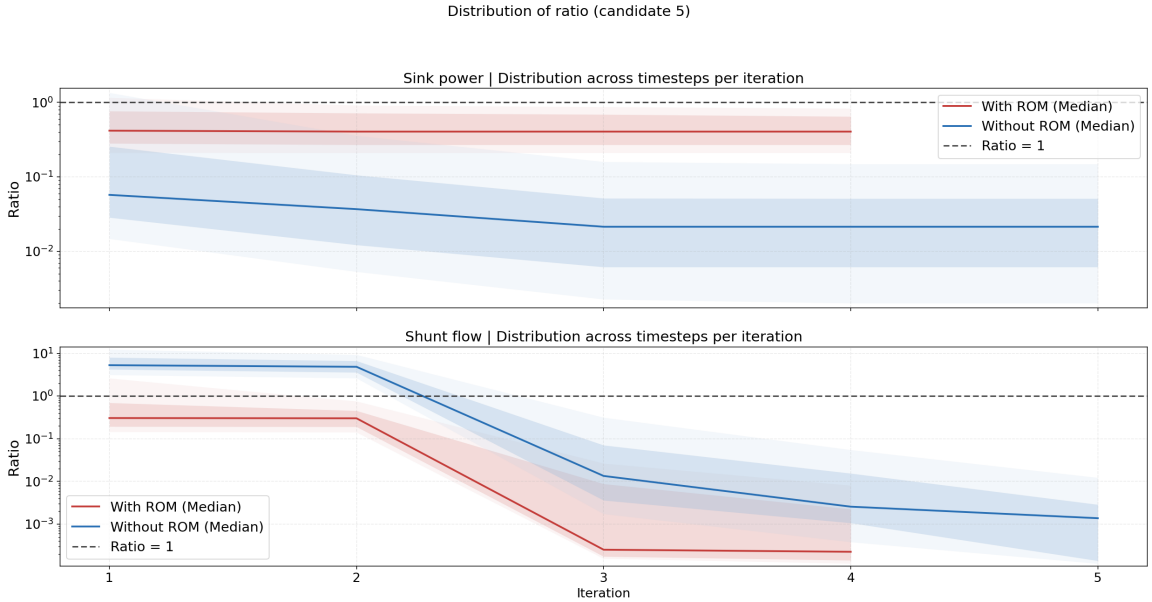


Figure B.6: Distribution of the convergence ratio for the sink power (top) and shunt flow (bottom) criterion across all time steps per iteration for candidate 5. Solid lines show the median ratio per iteration for the simulations with and without the ROM, while the shaded regions indicate the spread across time steps.

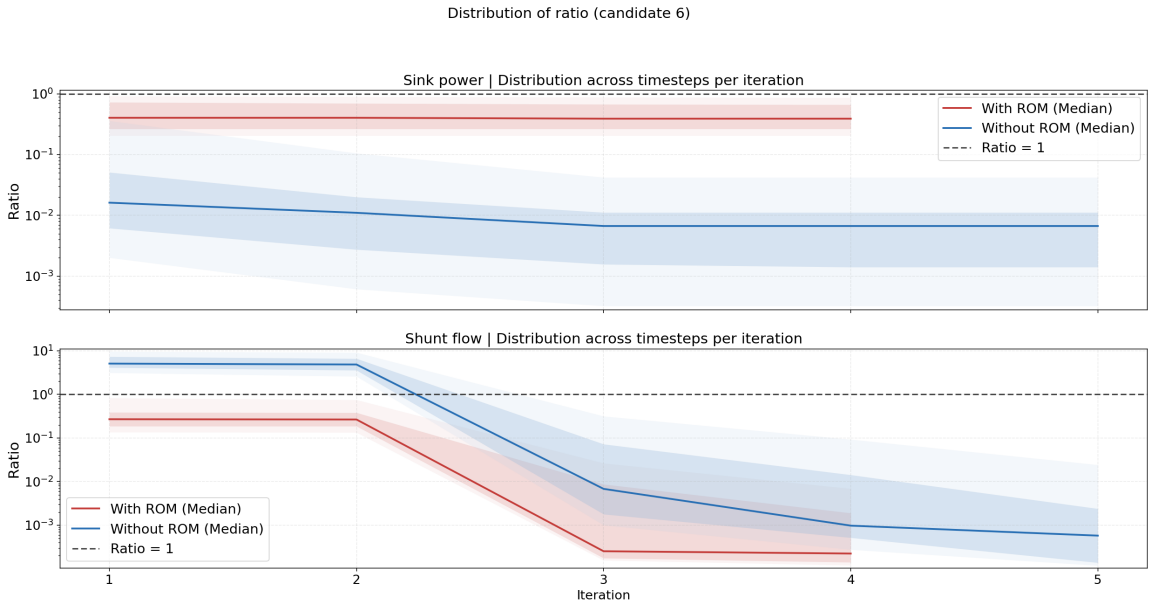


Figure B.7: Distribution of the convergence ratio for the sink power (top) and shunt flow (bottom) criterion across all time steps per iteration for candidate 6. Solid lines show the median ratio per iteration for the simulations with and without the ROM, while the shaded regions indicate the spread across time steps.

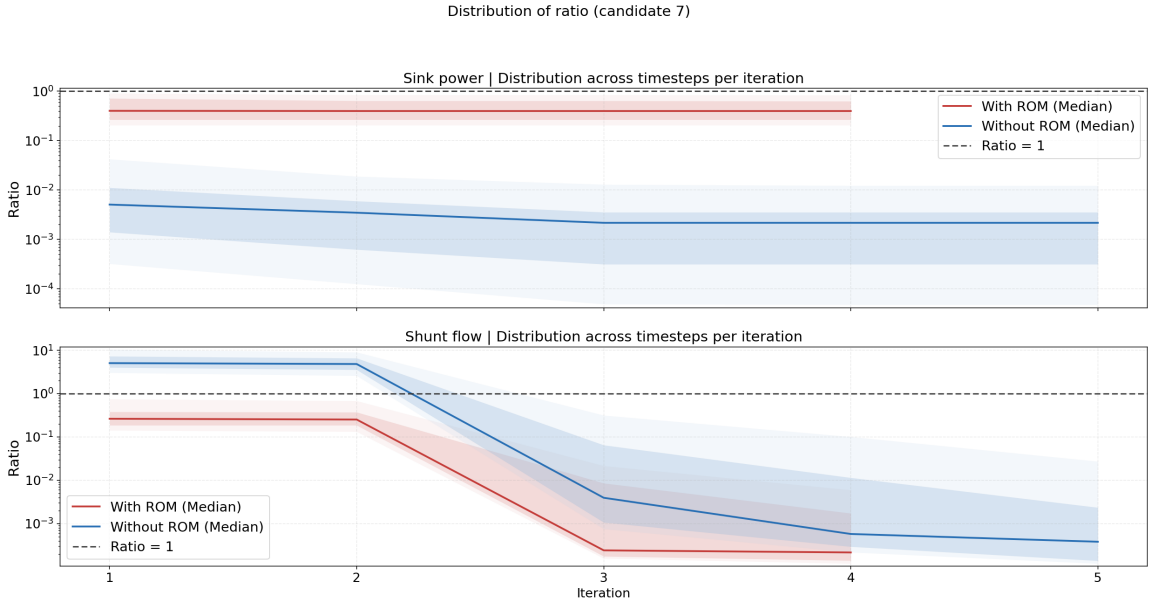


Figure B.8: Distribution of the convergence ratio for the sink power (top) and shunt flow (bottom) criterion across all time steps per iteration for candidate 7. Solid lines show the median ratio per iteration for the simulations with and without the ROM, while the shaded regions indicate the spread across time steps.

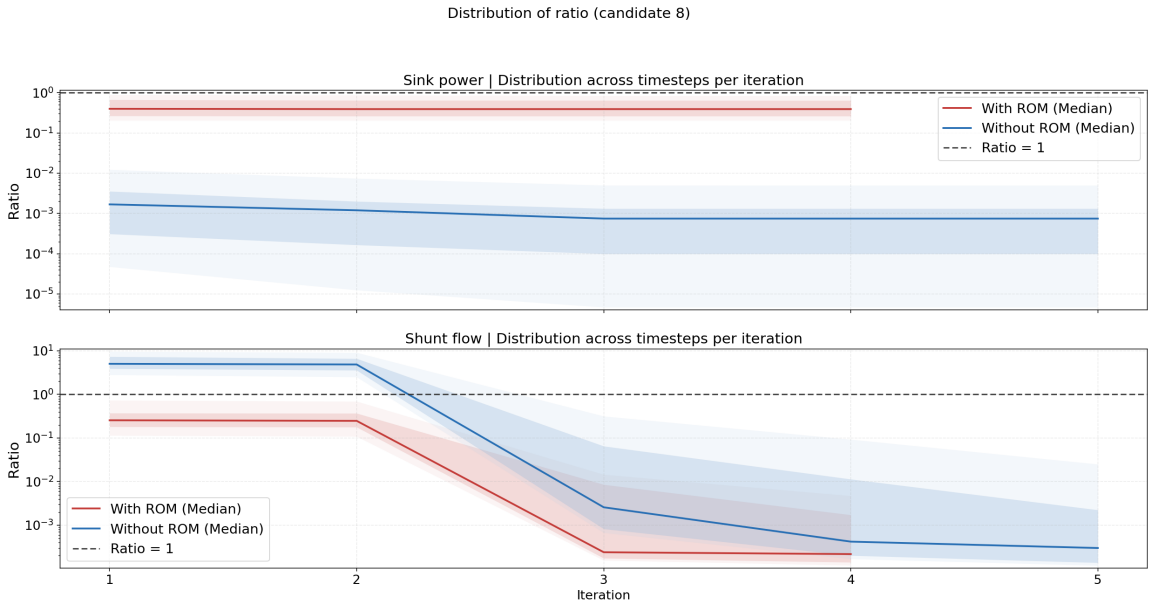


Figure B.9: Distribution of the convergence ratio for the sink power (top) and shunt flow (bottom) criterion across all time steps per iteration for candidate 8. Solid lines show the median ratio per iteration for the simulations with and without the ROM, while the shaded regions indicate the spread across time steps.

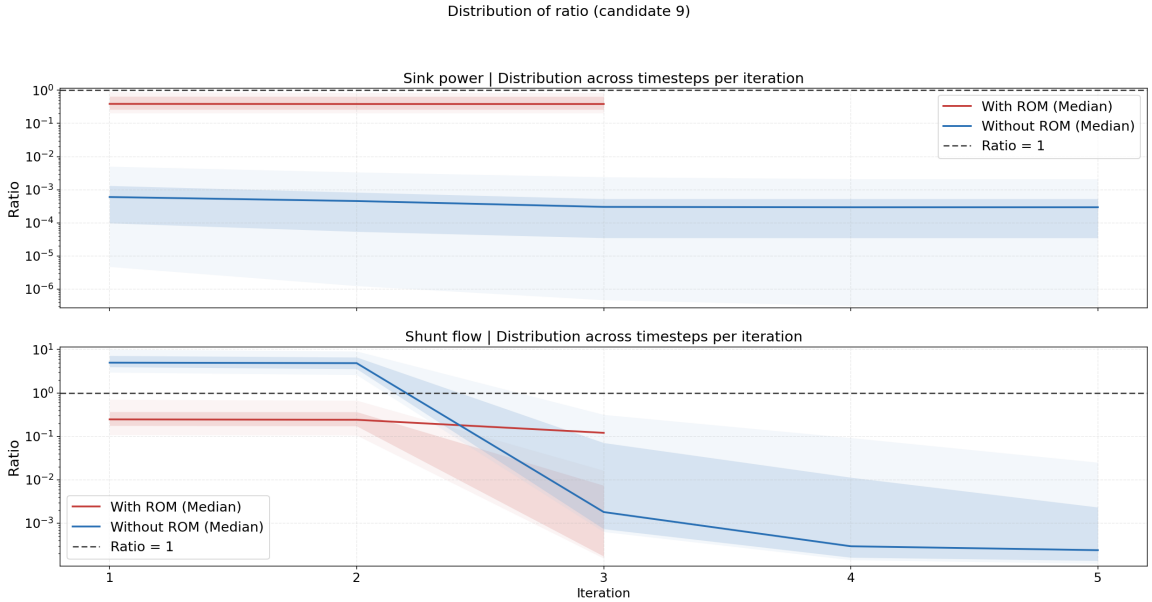


Figure B.10: Distribution of the convergence ratio for the sink power (top) and shunt flow (bottom) criterion across all time steps per iteration for candidate 9. Solid lines show the median ratio per iteration for the simulations with and without the ROM, while the shaded regions indicate the spread across time steps.

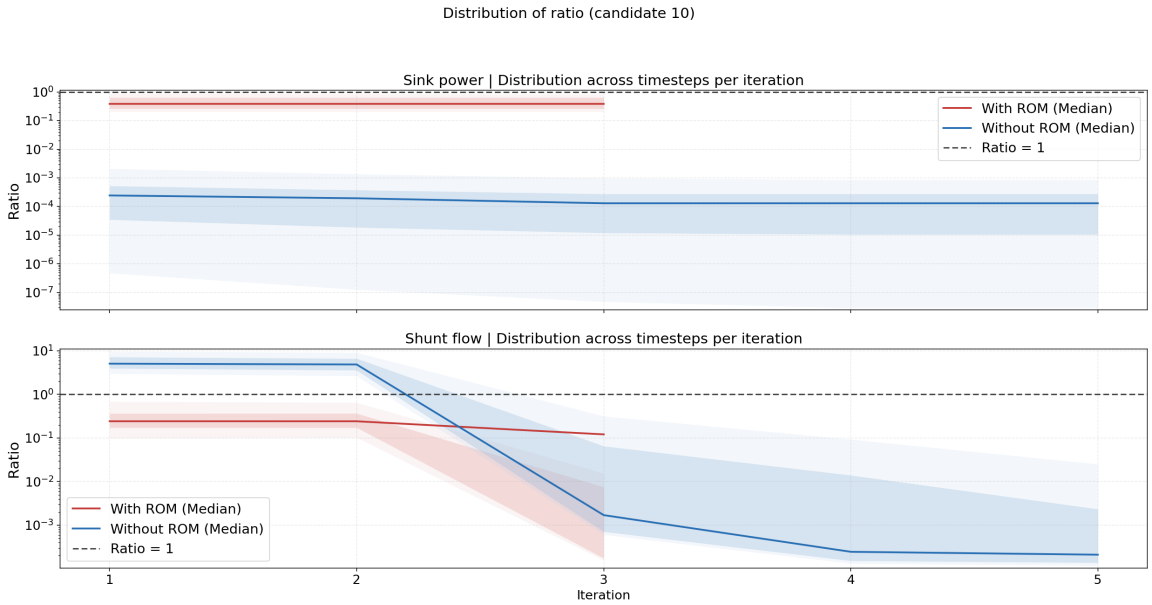


Figure B.11: Distribution of the convergence ratio for the sink power (top) and shunt flow (bottom) criterion across all time steps per iteration for candidate 10. Solid lines show the median ratio per iteration for the simulations with and without the ROM, while the shaded regions indicate the spread across time steps.

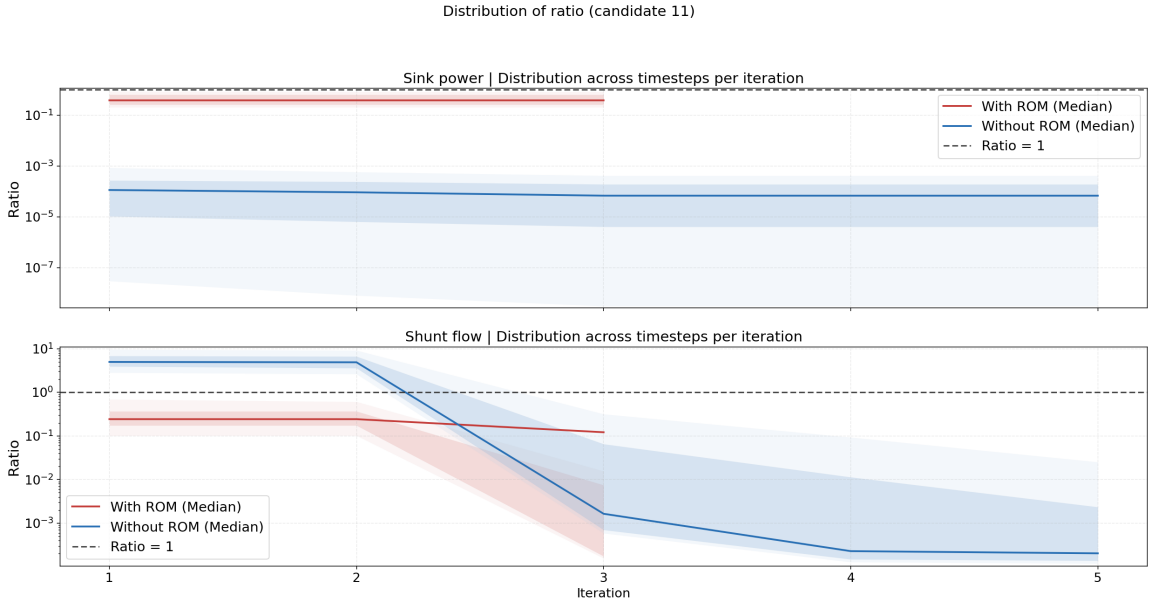


Figure B.12: Distribution of the convergence ratio for the sink power (top) and shunt flow (bottom) criterion across all time steps per iteration for candidate 11. Solid lines show the median ratio per iteration for the simulations with and without the ROM, while the shaded regions indicate the spread across time steps.

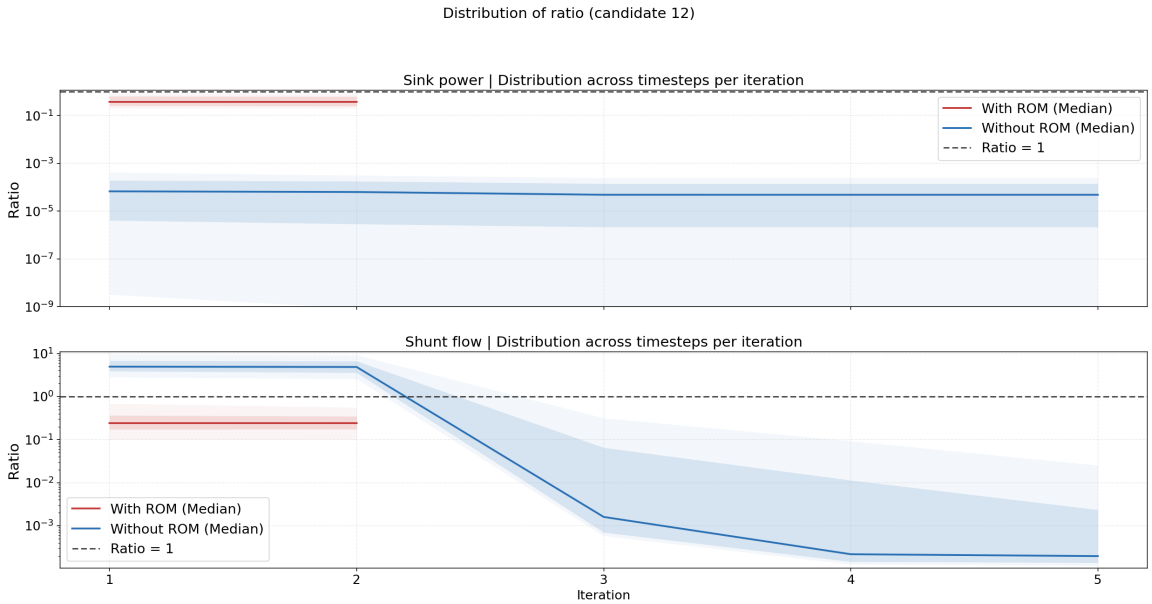


Figure B.13: Distribution of the convergence ratio for the sink power (top) and shunt flow (bottom) criterion across all time steps per iteration for candidate 12. Solid lines show the median ratio per iteration for the simulations with and without the ROM, while the shaded regions indicate the spread across time steps.