Data-Driven Modeling of Large-Scale Control Systems at the European Spallation Source (ESS)

Closed-Loop Modeling of the Cryogenic Moderator System (CMS) and the Target Helium Cooling System (THCS) Using Transformers and LSTMs

Eskil Olofsson

LUND
UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6313
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2026

Abbreviations

ARMAX Autoregressive Moving Average with Exogenous Variables

CMS Cryogenic Moderator System

EPICS Experimental Physics and Industrial Control System

ESS European Spallation Source

FFN Feed-Forward Network

GRU Gated Recurrent Unit

HiPPO High-Order Polynomial Projection Operators

IOC Input/Output Controller

ITS Iterative Training Strategy

LLM Large Language Model

LSTM Long Short-Term Memory

LTI Linear Time-Invariant

ML Machine Learning

MSE Mean Square Error

NARX Nonlinear Autoregressive with Exogenous Variables

NLP Natural Language Processing

PID Proportional–Integral–Derivative Controller

PV Process Variable

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

S4 Structured State Space Sequence Model

THCS Target Helium Cooling System

TMCP Target Moderator Cryoplant

Abstract

Control systems at European Spallation Source (ESS) are large-scale, complex and difficult to model using traditional approaches. Data-driven methods are therefore required to model systems such as the Cryogenic Moderator System (CMS) and Target Helium Cooling System (THCS). This work presents a control-system modeling approach based on the Transformer and Long Short-Term Memory (LSTM), enabling long-range multivariate predictions using control variables as exogenous inputs. The proposed framework integrates these models into a closed-loop setting, enabling iterative, autoregressive, predictions during both training and deployment. The results were promising for the THCS and artificial systems, while performance on the CMS was more limited. This discrepancy was likely caused by insufficient system excitation in the available datasets, owing to the absence of beam-on-target operations at ESS to date. While the proposed data-driven framework demonstrates considerable potential, its overall reliability is currently constrained by the limited diversity of operational data and a lack of out-of-domain testing.

Keywords: data-driven modeling, control systems, Transformers, LSTM, NARX

Acknowledgements

The project was conducted at the request of the European Spallation Source (ESS) and carried out at the facility. I would like to thank my supervisors at ESS: Karin Rathsman for great advisement and for making the project possible and Attilah Horvat for skilled system expertise. Also a thank you to Evan Foy for help with the THCS system. In addition, I would like to thank my colleagues at ESS, especially the other Master's thesis students for good company this spring.

The thesis was written with the Department of Automatic Control at Lunds Tekniska Högskola. Thank you to Johan Eker and Tore Hägglund for excellent academic guidance moving the project forward in the right direction. I would also like to thank my examiner, Bo Bernhardsson, for valuable input.

Contents

1

Introduction

The ESS, located in Lund, Sweden, is a next-generation neutron research facility designed to provide the world's most powerful long-pulse spallation neutron source. ESS (see Figure 1.1) enables a wide range of scientific investigations by using neutrons to examine the structure and dynamics of materials at scales ranging from macroscopic to atomic. Neutron scattering techniques supported at ESS are applied across disciplines including materials science, chemistry, biology, energy research and fundamental physics [1]. The facility currently hosts 15 state-of-the-art instruments optimized for different experimental needs, such as: *HEIMDAL* for diffraction and imaging; *BIFROST* for high-resolution spectroscopy of quantum materials; and *BEER* for engineering diffraction and residual stress analysis [2].

By accelerating protons in superconducting RF cavities, the beam will reach a power of $5MW$, with $2GeV$ protons reaching speeds of 96% of the speed of light [3]. Initial operation will be at a lower effect of $2MW$. By firing bursts of protons at $14Hz$ and colliding them with a tungsten target, ESS generates a neutron beam through spallation. This long-pulse neutron beam will be collimated and used in the instruments around the target.

ESS relies on several cryogenic systems for its operation. In the accelerator, there is a helium cryogenic system used for cooling the superconducting cavities to $2K$. At the target, the CMS plays a critical role in the thermalization of the neutrons and the THCS cools the target wheel spinning the tungsten target.

The CMS utilizes liquid hydrogen to thermalize the spallated neutrons, since hydrogen has a high neutron scattering cross section. The thermalized, slower, neutrons can then be used for research because they, in turn, have a higher scattering cross-section in the target analyzed. The spallation process induces significant heat loads on the target. In the CMS this leads to

temperature and pressure spikes in the system that is operated at around $17K$ and $11bar$. In the target, the spallation induces a high heat load, not only on the CMS, but also on the target wheel, as well as the surrounding apparatus.

There is thus a need for robust cooling systems that can handle these heat loads. In the CMS, the hydrogen is cooled through a heat exchanger connected to the Target Moderator Cryoplant (TMCP). The TMCP circles liquid helium back to cooling compressors and operates in a steady state to get a desired temperature lower than the warmed up exchanger return temperature. The target wheel is cooled by a separate cooling system, the THCS, which relies on circulating helium gas in the wheel. The helium is cooled by several water heat exchangers.

Figure 1.1 *An aerial picture of ESS taken by Skanska. In the background, extending toward the top left corner, is the proton linear accelerator. The middle building is the target building. The circular, sector-shaped building is the long instrument building. In front of the target building, towards the moat, the office buildings are visible.*

1.1 Problem formulation

The cryogenics and cooling systems are large-scale control systems relying on several Proportional–Integral–Derivative Controllers (PIDs) for its control. In the CMS, this is done by control operatives who, based on experience and calculation, slowly ramp desired setpoint values to bring the system into its cooling mode. This empirical approach demands experienced operators. The CMS is especially critical, as hydrogen is both a safety hazard and difficult to

control due to phase transitions caused by heat loads. The system must therefore be operated within strict protocols, making it difficult to perform system identification experiments that reveal detailed system behavior. Obtaining a model of the system would possibly provide insights into system dynamics, enable operator training through simulation and allow new control strategies to be evaluated without risking the live system.

Modeling the full system is difficult for several reasons. For one, it is highly dimensional, with many different Process Variables (PVs) interacting. It also suffers from time delays; for example, it takes several minutes for cooling helium from the TMCP to reach the heat exchangers at the CMS, due to it being located several hundred meters away. Another issue is the complexity of the system: the cryogenic properties are non-linear and the entire system operates in several different cryogenic regimes. This rules out some classical linear modeling approaches. Obtaining a first-principles model capturing the intricate dynamics is a seemingly impossible task.

There is thus a need for a data-driven modeling approach. In the time of an emerging "AI revolution", these models have been shown to yield good results in modeling language tasks, which are inherently very complicated. These models have shown to be able to encode context, process sequential information and capture complex relationships. These tools do not only exist for Large Language Model (LLM) but are also applicable to time-series data. Is it, however, possible to apply them in a control system context?

1.2 Purpose and Research Questions

The purpose of this study is to investigate the feasibility of data-driven modeling for complex, large-scale, control systems such as the CMS and the THCS, to establish a systematic methodology for their development and evaluation. To this end, the study addresses the following research questions:

- How can a modeling pipeline be designed for data-driven modeling of large-scale control systems?
- Which machine learning methods are suitable for modeling the dynamics of complex control systems such as the CMS and the THCS?
- To what extent is it possible to develop data-driven models of the CMS and the THCS and are limitations imposed by data availability, inherent system properties, or the capabilities of the models tested?

1.3 Work Overview

The study began with a literature review to establish the theoretical and methodological foundations of the work. Relevant research on data-driven modeling, time-series forecasting using Machine Learning (ML) methods, deep learning for prediction, nonlinear control system modeling and data-driven system identification was examined. State-of-the-art publications were surveyed to identify effective modeling strategies, neural network architectures and evaluation practices used in comparable applications, while also highlighting the limitations of traditional approaches and motivating the use of modern data-driven ML methods.

A modeling strategy was then defined based on the general system identification framework from Ljung [4]. First, system PVs, sampling frequency and other modeling decisions were selected together with system experts. Data was subsequently collected from the `EPICS ARCHIVER`.

Several candidate ML models were then selected based on insights from the literature review, namely LSTM and Transformer architectures.

A strategy for prediction, training and validation was also developed to improve performance over autoregressive iterative predictions. Rather than training only on one-step predictions, the models were trained to predict several steps ahead. Since the systems exhibited strong persistence, the main performance criterion was improvement over the naive predictor assuming constant values.

Finally, validation was carried out both using the training criterion and by evaluating performance on generated artificial systems, where data availability was not a limiting factor.

1.4 Modeling

The selected models were LSTM and Transformer architectures, implemented as one-step predictors operating autoregressively over longer prediction horizons. The general formulation of the closed-loop system model is:

$$\begin{aligned} X_{k+1} &= f(X_{k-m:k}, U_{k-m:k}) \\ U_k &= c(X_{k-m:k}, U_{k-m:k-1}, r_k) \end{aligned}$$

where f is the predictive model (an LSTM or Transformer variant) and c is a control law. $X_{k-m:k}$ denotes the history of endogenous target variables,

$U_{k-m:k}$ represents the history of exogenous control inputs, and r_k is the reference signal.

Since the models need to perform over long iterative prediction sequences, they were trained using the proposed **Iterative Training Strategy (ITS)** (see Section 4.2.1). The ITS trains the models on multi-step autoregressive predictions, teaching them to follow observed trajectories while using control inputs taken from the data. The idea can be roughly summarized as (with $\hat{X}$ denoting a prediction):

$$\begin{aligned}
\hat{X}_{k+1} &= f(X_{k-m:k}, U_{k-m:k}) \\
\hat{X}_{k+2} &= f(\{X_{k-(m-1):k}, \hat{X}_{k+1}\}, U_{k-(m-1):k+1}) \\
&\vdots \\
\hat{X}_{k+n} &= f(\{X_{k-(m-n):k}, \hat{X}_{k+1:k+(n-1)}\}, U_{k-(m-n):k+(n-1)})
\end{aligned}$$

where a prediction is appended to the model history together with observed control data and the loss is then quantified as the difference over the entire sequence.

By training on longer trajectories, which is only possible using ITS, the idea is to encourage models to make more modest one-step predictions to accommodate longer horizons, as the loss backpropagates through all predictions. Adapting the models for longer horizons essentially makes the strategy act as a regularizer. In this way, the hope is that the models learn the true system behavior and improve long-horizon prediction performance.

1.5 Brief Overview of Results and Discussion

Because the systems at ESS were relatively new at the time of this study, data availability was a significant challenge. This was particularly true for the CMS, which had only five operational cooldowns consisting mostly of steady-state data that did not sufficiently excite the system. Furthermore, beam-on-target operations had not yet started, limiting the variety of operational modes for both the CMS and the THCS.

Utilizing data-driven LSTM and Transformer models to predict multiple PVs simultaneously showed promising results, particularly for the THCS. Although the THCS dataset contained fewer total data points, a larger portion featured meaningful variation. Consequently, the THCS models performed significantly better than the baseline, producing predictions reliable and consistent enough with realistic system behavior.

In contrast, due to a lack of sufficiently varied data, the CMS models performed only slightly better than the naive predictor on average, with simulated sequences occasionally diverging from the true trajectories. Specifically, the LSTM achieved the best results for the THCS, whereas the Transformer captured the temporal dependencies in the CMS better than the LSTM. Overall, for a given model size, the two architectures demonstrated comparable performance.

Evaluation on artificial systems and the ITS showed good results for all models, especially over longer prediction horizons, strengthening that the data-driven modeling approach seems promising. Beyond data availability, a key limitation of this study is that it did not explore out-of-domain validation, such as evaluating the models under novel operating scenarios or entirely new control laws.

For better performance and more rigorous validation, more operational data will be required. This data will soon be available as ESS starts up beam-on-target operation.

1.6 Novelty and Contributions

The main novel aspects and contributions explored in this thesis can be summarized as follows:

- This thesis explores a modern data-driven approach in a control systems setting. Many traditional methods exist for modeling control systems, while many modern approaches have been developed for predicting purely time-series data or language tasks, such as the Transformer and the LSTM. However, applying these modern data-driven methodologies in a closed-loop control systems setting was relatively unexplored and similar projects were difficult to find in the literature.
- The closed-loop setting also imposed practical restrictions on how the models could be designed and implemented. The models needed to operate as autoregressive one-step predictors while handling multivariate target variables (endogenous variables) together with control variables (exogenous variables). This made Nonlinear Autoregressive with Exogenous Variables (NARX) models well suited for the problem. Additionally, the scale of the problem and the simultaneous multivariate predictions motivated the use of these modern architectures.
- Finally, the thesis proposed the Iterative Training Strategy (ITS) to improve multi-step prediction performance. Although the closed-loop

setting required a one-step predictor, reliable multi-step performance was necessary for simulation purposes. By using observed control data and letting the models make autoregressive predictions that could be compared with real trajectories, the loss could be backpropagated through the entire prediction chain. This encouraged the models to make more modest one-step predictions that remained stable over longer trajectories. Models trained using the ITS showed better performance, particularly for longer prediction horizons.

2

System Overview

There are several cooling systems acting on the ESS Target, as shown in Figure 2.1. The Target Monolith Vessel is a vacuum-sealed structure designed to encapsulate the target, protecting the surrounding components from radiation and mechanical damage. Because the vessel operates under vacuum, there is no convective heat dissipation; consequently, all internal components must be actively cooled to withstand the heat generated by the proton beam and the spallation process.

The Target Helium Cooling System (THCS) provides cooling for the target wheel. The moderator water cooling system cools the moderator blocks and the shielding and plugs water cooling system cools the shielding and the plugs. The Cryogenic Moderator System (CMS) cools the spalled neutrons themselves, slowing them down to the desired energy range. All of these systems must be capable of handling the high heat loads resulting from beam operation and spallation, with the CMS being the most critical due to its cryogenic operation and sensitivity to thermal disturbances.

The Target Moderator Cryoplant (TMCP) is a connecting subsystem of the CMS that supplies liquid helium for cooling the CMS. In practice, the TMCP and CMS form a single system, but they are distinguished by name because they are located far apart and the TMCP is operated separately under steady-state conditions. Although the TMCP also contains operator-adjustable control valves, these are adjusted solely to support the operation of the CMS.

2.1 CMS

A: The Cryogenic Moderator System (CMS), seen in Figure 2.1 and in more detail in Figure 2.2, consists of several parts. Closest to the target is the moderator. It consists of blocks where liquid hydrogen flows in and

thermalizes the neutrons. To even out the heat load from the periodically on-off beam there exists a heater, "EH-82650" (shown in Figure 2.2). The heater can also be used for applying experimental heat loads on the system. It sits at the top of the CMS Distribution Box.

B: The second part is the ColdBox. It has two tanks. In one of them, the buffer tank, liquid hydrogen is held at a certain level where vaporization occurs, both spontaneous and heat-induced, to counteract rapid pressure changes. Heat-induced vaporization is achieved using four external heaters acting on the tank. The vaporized hydrogen is then cooled by the smaller heat exchanger when the valve "CV-62029" (shown in Figure 2.2) is opened. This valve acts as a pressure controller and is operated by a PID acting on the CMS system pressure, measured by "PT-62073" (Fig. 2.2). The liquid height can be derived from a pressure difference measurement – "PDT-62067" Fig. 2.2).

The other tank is a converter for the spin isomers of hydrogen. To obtain good moderator performance, the fraction of parahydrogen needs to be higher. Therefore, there exists a converter for this. This has no effect on the cryogenic system itself.

Hydrogen is circulated in the CMS by two pumps, "P-62010" and "P-62011" (Fig. 2.2). The flow of the system, as a function of pump speed, but also temperature and pressure is measured by flow transmitters and are combined as "Total flow" in Figure 2.2. The corresponding PV is named "PLC-001".

C: At the Coldbox there are also heat exchangers. The main one circulates most of the liquid hydrogen to be cooled by liquid helium (see "TT-62085", "TT-62087", "TT-61300" and "TT-61500" in Figure 2.2). There is also a smaller heat exchanger that cools gas coming from the top of the buffer tank (see "TT-62117", "TT-62119" and "TT-61700" in Figure 2.2). Connecting to the heat exchangers is the TMCP, which circulates and cools the supplied liquid helium.

D: The TMCP is operated in a steady state, with a fixed temperature delta, ΔT, between the incoming helium and the returning cooled-down helium. The distance between the TMCP–CMS heat exchanger is several hundred meters which both introduces a time delay (of roughly 10 minutes) and a temperature drop off. Therefore, the helium is cooled to a lower than desired temperature before being transported to the CMS. At the TMCP Jumper Spool Box (JSB in Figure 2.2), two feed-forward injection valves are installed to inject warm helium into the pipelines, compensating for temperature

drops along the pipes and for disturbances induced by changes in the CMS ("CV-51880" and "CV-51870" seen in Fig. 2.2).

The CMS is operated mainly through three PID controllers acting on individual measurements. The feed temperature controller acts on control valves "CV-31840" and "CV-31450" (not shown in Fig. 2.2) which regulate the desired feed temperature "TI-31610" (not shown in Fig. 2.2 either). The return temperature controller operates a single valve, "CV-51870" (in Fig. 2.2), which injects warm helium to stabilize the return temperature to the TMCP inlet – "TI-512001" (Fig. 2.2).

Then there is the TMCP HP/LP pressure controller which controls the desired pressure in the system. The desired ratio between the high pressure line and the low pressure line is 4.1. The HP/LP controller ("PC-217001") regulates the specified pressure, "PI-217001", through valves ("CV-21600", "CV-21700", "CV-21650" and "CV-21750", neither of which are shown in Fig. 2.2). Finally, there is the pressure controller, acting on "CV-62029", previously mentioned and "CV-42310", which injects new hydrogen into the system (both shown in Fig. 2.2).

Operators guide the system by updating controller setpoints. To initiate a cool-down, for example, temperature targets are slowly ramped down under strict observation. Any deviation prompts a pause to allow the system to catch up. Because the system is critical and highly volatile, this conservative approach is necessary, resulting in frequent pauses and multi-day cool-down periods.

There are a lot more control loops than the mentioned ones, most operated around a pre-set value and are not part of the active control of the CMS. Most of these are mainly a part of the TMCP, which always operates in a steady state with a fixed ΔT. These control loops can be seen as a part of the system dynamics. Other valves are operated manually and used mostly during maintenance or set at specific values, not affecting control of the system. Several of them can be seen in Figure 2.2 (e.g. "CV-51850" or "CV-82012").

2.2 THCS

A: The Target Helium Cooling System (THCS) is a cooling system designed to remove heat loads from the target wheel, drive and shaft. The scheme of the THCS can be seen in Figure 2.3. Cooled helium gas is circulated through the entire unit, down to the wheel (see A in Figures 2.3 & 2.1), to remove the heat load caused by the proton beam and the subsequent spallation.

B: The helium is then circulated back and cooled by a water cooling system utilizing cold water from the Conventional Cooling Water Supply. The cooling of the helium gas is performed through three heat exchangers (see e.g. "TT-002", "TT-003" and "TT-005" in Fig. 2.3). The helium is circulated through two of them; after that, it is pressurized by a compressor ("SC-002" in Fig. 2.3), which raises the temperature of the gas ("TT-154" in Fig. 2.3). A final heat exchanger then cools the helium before it is circulated back through the target wheel. The system is operated by a manual setpoint for the compressor ("SC-002").

C: Desired temperature setpoints for the helium, "TT-003", "TT-004" and "TT-005" (seen in Fig.2.3), are then set, which the pumps "SICA-312", "SICA-212" and "SICA-112" (shown in Fig.2.3, but no labels) act on to regulate the cooling to the desired level through PID loops. In order to obtain the correct helium temperature, the feed water temperature to the heat exchangers, measured by "TT-316", "TT-216" and "TT-116" (shown in Fig.2.3), must also follow a desired setpoint. This is regulated by the valves "TCV-316", "TCV-216" and "TCV-116" (shown in Fig.2.3, but no labels).

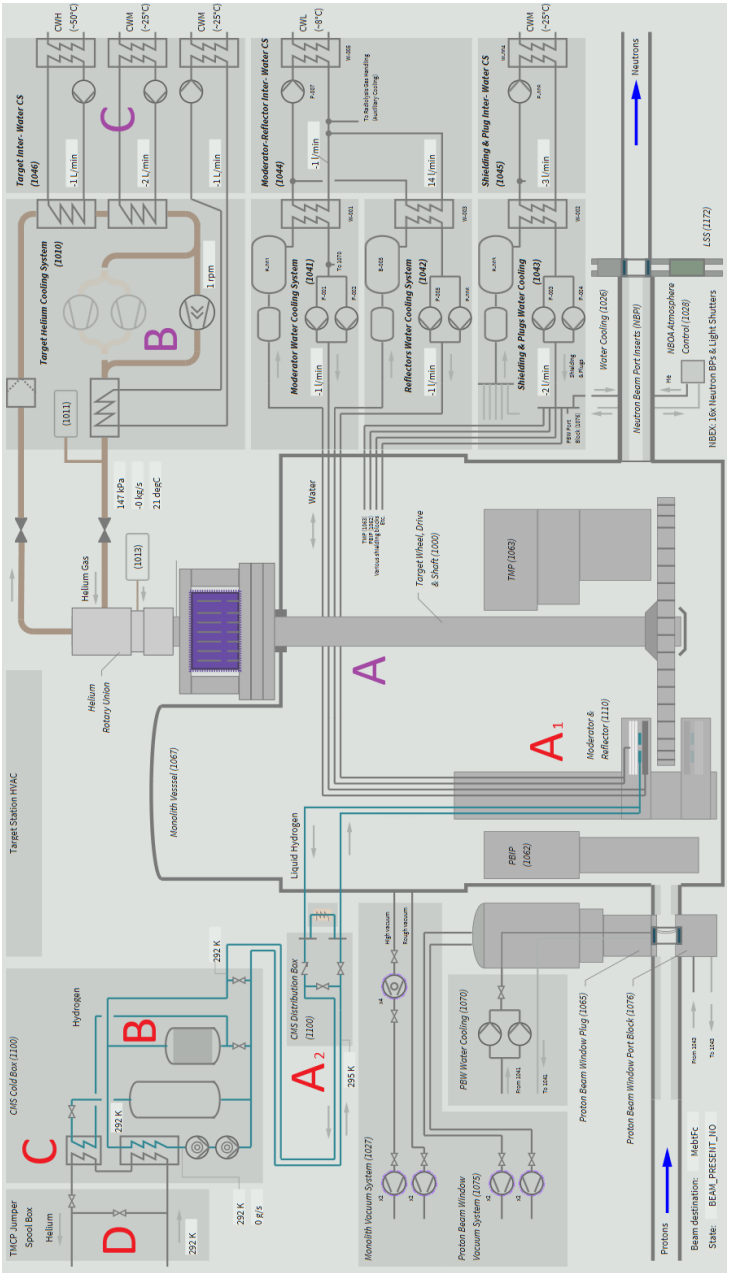


Figure 2.1 *The target at ESS includes several cooling systems. Using CS-Studio a visual schemes of the cooling systems has been made. In the upper left corner, a brief scheme of the CMS (in red) is shown and in the upper right, the THCS (in purple). Letters denotes parts to be explained in the text.*

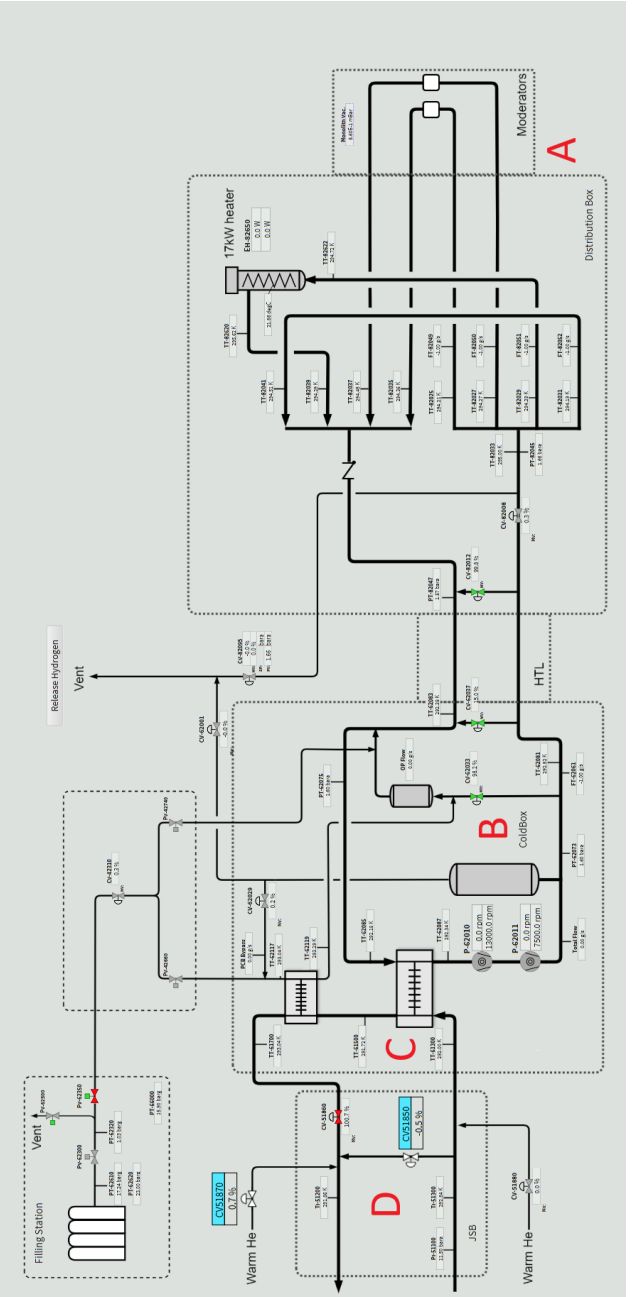


Figure 2.2 *A scheme of the CMS showing PVs and components. The three main parts of the system can be seen, as they are separated by the dashed lines. A shows the moderator and the Distribution Box, B the Coldbox, C the heat exchangers at the Coldbox and D the connecting part to the TMCP, the Jumper Spool Box.*

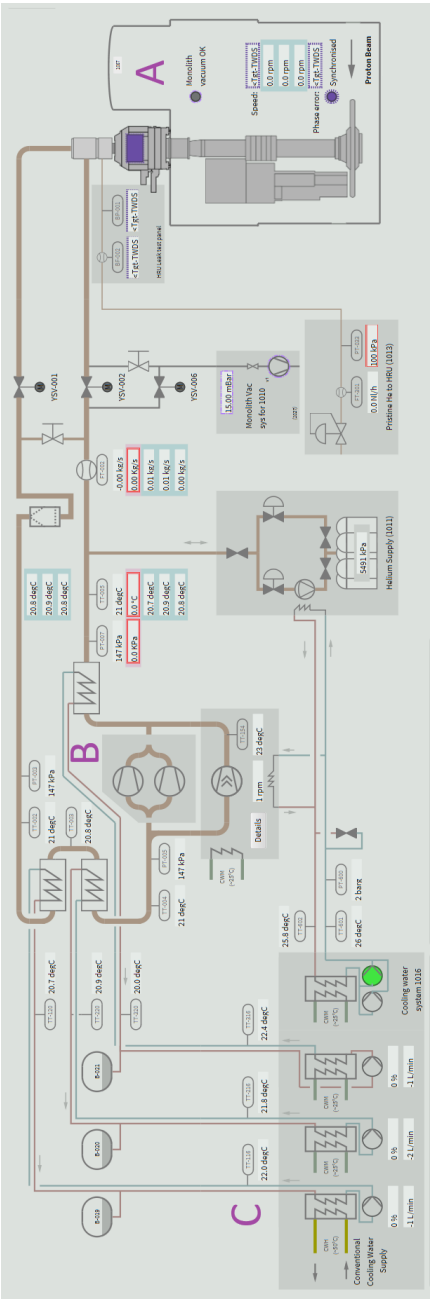


Figure 2.3 *A scheme of the THCS showing PVs and components. A is the target wheel, B the helium heat exchangers and C the Conventional Cooling Water Supply.*

3

Theory

Dynamical systems are commonly described using state-space representations. Since physical equations are formulated in continuous time, it is natural for systems derived from physical equations to also have a state-space formulation in continuous time. A general description of a non-linear continuous-time system can be written as [4]

$$\begin{cases} \dot{x}(t) &= f(t, x(t), u(t), w(t); \theta) \\ y(t) &= h(t, x(t), u(t), v(t); \theta) \end{cases}$$

where $x(t) \in \mathbb{R}^n$ denotes the system state, $u(t) \in \mathbb{R}^m$ the input and $y(t) \in \mathbb{R}^p$ the measured output. The signals $w(t)$ and $v(t)$ represent process disturbances and measurement noise, respectively and $\theta \in \mathbb{R}^d$ denotes model parameters.

In practice, measurements and control actions are implemented digitally and are therefore available only at discrete sampling instants. Consequently, system identification and data-driven control are naturally formulated in discrete time. The general finite-dimensional discrete-time system on state-space form can be expressed as [4]:

$$\begin{cases} x_{k+1} &= f(k, x_k, u_k, w_k; \theta) \\ y_k &= h(k, x_k, u_k, v_k; \theta) \end{cases}$$

In most control applications, the explicit dependence on k is absent, defining a causal or time-invariant system.

Models can be derived from physical laws, first-principles, commonly referred to as white-box modeling. When the model structure is based on physical insight but some parameters are estimated from data, the approach

is known as gray-box modeling. The defining characteristic of these categories is that the underlying physical processes are known and that there is explicit insight into how the system operates [4, 5].

However, deriving such models is not always feasible due to complexity, uncertainty, or incomplete knowledge of the system. In such cases, alternative approaches are employed, including linear black-box models such as Autoregressive Moving Average with Exogenous Variables (ARMAX) models. More complexity and non-linear relationships demands more flexible, sophisticated, approaches, such as deep learning models [4, 5].

In a black-box model, f can be interpreted either as an encoder mapping the system into a latent state-space representation, with h acting as a decoder to the target (measurement) space, or f can be taken to represent the entire input–output model, in which case $y = x$. This distinction is mostly a matter of how the state space is imagined. Since ML models typically encode information into several latent-space representations, the only states that carry interpretable meaning are the observable states in the target space. In this case, it is often simpler to completely disregard the measurement equation h and focus solely on the model dynamics:

$$\begin{cases} x_{k+1} &= f(k, x_k, u_k, w_k; \theta) \\ y_k &= x_k \end{cases}$$

Back to the full state-space representation; for many practical systems, the non-linear dynamics can be well approximated by linear models near a given operating point. Linearizing the system around such a point results in a Linear Time-Invariant (LTI) discrete-time state-space model of the form [4]:

$$\begin{cases} x_{k+1} &= Ax_k + Bu_k + w_k \\ y_k &= Cx_k + Du_k + v_k \end{cases}$$

where A, B, C and D are constant system matrices. This LTI approximation provides a compliant mathematical framework that enables the application of classical and robust control methods, such as stability analysis, controller synthesis using pole placement, modeling of uncertainty and performance evaluation.

3.1 System Identification

In system identification, a white-, grey- or black-box model is not only required to describe the system dynamics, but also to specify how future

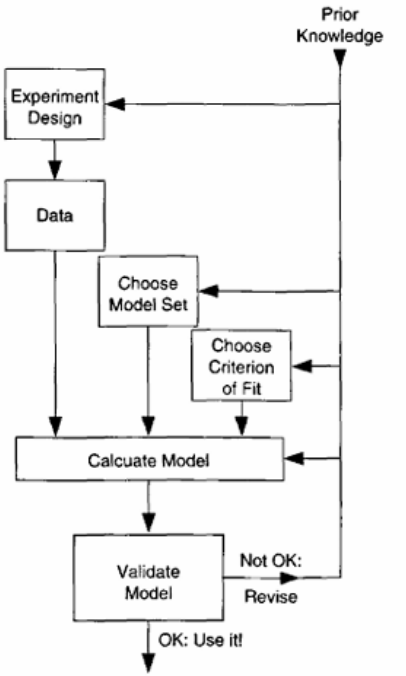


Figure 3.1 *Algorithmic procedure for model building defined by Ljung [4]. Picture taken from Ljung as well.*

outputs are predicted. To achieve this, the model-building framework can be viewed as an algorithmic procedure [4, 6] as can be seen in Figure 3.1. This scheme can be explained by four major components: **Data Collection**, including data and experiment design; **Model selection**, including the block choose model set; **Training**, including Criterion of fit and Calculation; and **Validation** [4]. The framework is in general terms described below:

Data Collection: Let:

$$\mathcal{D} := \{(u_0, x_0), (u_1, x_1), \ldots, (u_N, x_N)\}$$

denote a data set collected from the system. This data must be informative enough to cover the relevant operating conditions of the system and contain sufficient excitation to reveal the underlying input-output dynamics. In ideal cases, this is achieved in open loop by applying well-designed excitation signals (that is, usually step inputs) and observing the resulting system response.

In practice, however, data is often collected while the system operates in closed loop, since the plant may be unstable or too critical to run in open loop. Under feedback, the input u_k is partly determined by the output x_k, which can reduce the information content of the data. To ensure that the system can be identified, additional external excitation must be injected through the controller. In particular, the reference signal r_k must be sufficiently exciting so that the plant dynamics can be distinguished despite the presence of feedback [4].

The user must also decide which signals to measure and how to classify them as inputs or outputs.

Model Selection: Next, the set of model candidates needs to be selected. A model $\mathcal{M}(\theta)$ is defined as a parametric predictor:

$$\mathcal{M}(\theta): \quad \hat{x}_{k+1}(\theta) = f(U_{k-m:k},\, X_{k-m:k}\,;\theta)$$

Where $U_{k-m:k},\, X_{k-m:k} = \{x_k, u_k ... x_{k-m}, u_{k-m}\}$ is a collections of past inputs and $f(\cdot;\theta)$ is a mapping parameterized by θ. The models can range from white- or grey-box models derived from physical equations to black-box models such as ARMAX or more sophisticated ML approaches.

Training: The goal of system identification is to learn the parameters that accurately reproduce the system's input–output data. This requires defining a mapping, $\mathcal{A}$, that computes the optimal parameters θ from the available dataset:

$$\mathcal{A}: \ \mathcal{D} \longmapsto \hat{\theta}_{\mathcal{D}}$$

where $\hat{\theta}_{\mathcal{D}}$ is chosen to minimize a suitable criterion of fit evaluated on the dataset. This criterion could, as is common, be Mean Square Error (MSE) or Maximum Likelihood. The mapping $\mathcal{A}$ is in the case of MSE be defined as:

$$\mathcal{A}_{MSE}: \quad \hat{\theta}_{\mathcal{D}} = \arg\min \frac{1}{N} \sum_{n=1}^{N} \left(x_n - \hat{x}_n(\theta)\right)^2$$

Validation: Summarizing the learning formulation, the goal is to choose a model $\mathcal{M}(\theta)$ and a learning criterion $\mathcal{A}$ that allow the model to generalize from a given dataset $\mathcal{D}$. That is, $\mathcal{M}(\theta)$ should approximate the true system behavior well enough to both interpolate known data and extrapolate to unseen data. The final model's performance is then verified using chosen validation criteria, such as residual analysis and prediction accuracy on a validation dataset. A model is considered acceptable only if it adequately captures the system dynamics within the intended operating range.

3.2 Machine Learning Methods

Modeling the system's underlying dynamics requires capturing highly non-linear behavior characterized by multivariate dependencies and significant temporal lags. Classical approaches, such as multivariate ARMAX models or linear regression, often fall short in this environment as they lack the capacity for complex non-linear mapping. While static deep learning architectures like Feed-Forward Networks (FFNs) can approximate non-linear functions, they ignore the sequential ordering and temporal context of the data.

To resolve these limitations, the LSTM architecture utilizes gated memory states to manage both long- and short-term dependencies [7]. Currently, however, the Transformer architecture has emerged as the state-of-the-art for modeling sequential data, especially for Natural Language Processing (NLP) tasks. By replacing recurrence with attention mechanisms, the Transformer explicitly maps correlations across historical data, enabling more efficient processing of long-range dependencies [8].

These architectures form the basis for most sequential learning tasks. A theoretical understanding of their internal mechanics is necessary to contextualize both the objectives of this work and the contributions of related studies. Consequently, a technical overview of these learning frameworks is provided in the following sections.

3.2.1 FFN

A Feed Forward Neural Network (FFN) is defined as:

$$\begin{aligned} \mathrm{FFN}(x) &= f_L(\dots f_2(f_1(x))) \\ f_i(x) &= \sigma(W_i x + b_i) \end{aligned}$$

Where L denotes the number of layers, σ an activation function, often a Rectified Linear Unit (ReLU):

$$\mathrm{RELU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

and $W_i x + b_i$ is an affine transformation. The maps generated for each layers are allowed to have different dimensions as long as the input and output of each layer fits the previous and the next.

3.2.2 LSTM

Recurrent Neural Networks (RNNs), unlike FFNs, are a family of neural networks that feed back outputs of parts of the system to the next input,

to include previous information in the next prediction [9]. This enables sequential processing of data. It is done by encoding a latent space that captures information from past sequences in the model. In this way, these architectures work by updating their encoding through an entire sequence, with the model learning to regulate the flow of information.

LSTM networks are a type of RNNs designed to model sequential data by maintaining both a hidden state h_t (short-term memory), which is updated iteratively over time and a cell state c_t (long-term memory). This aims to solve the vanishing and exploding gradient problem common in RNNs, which essentially means that the loss cannot propagate back correctly to update information flow early in a sequence. By using both a current hidden state and a cell state, the LSTM loss can more directly influence long-term behavior. Given an input sequence:

$$X \in \mathbb{R}^{T \times d_{\text{in}}}$$

where T is the sequence length and d_{in} is the input feature dimension. The LSTM processes the sequence one time step at a time, at each time step, t, the LSTM updates its hidden state and its cell state:

$$h_t, c_t = \text{LSTM}(x_t, h_{t-1}, c_{t-1}), \quad h_t, c_t \in \mathcal{Z}^{d_{model}}$$

where $x_t \in \mathbb{R}^{d_{\text{in}}}$ is the input at time step t. The forward pass of an LSTM cell

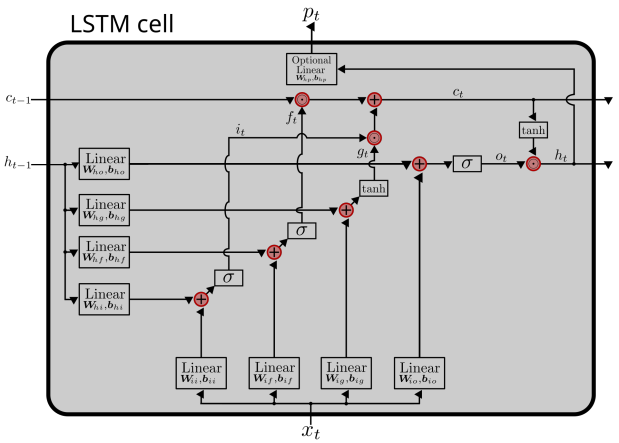


Figure 3.2 *Flow Scheme of an LSTM cell. By Leouscin, CC0, via Wikimedia Commons [10]*

is given by [11, 7]:

$$
\begin{aligned}
f_t &= \sigma(W_{if}\, x_t + b_{if} + W_{of}\, h_{t-1} + b_{of}) \\
i_t &= \sigma(W_{ii}\, x_t + b_{ii} + W_{oi}\, h_{t-1} + b_{oi}) \\
o_t &= \sigma(W_{io}\, x_t + b_{io} + W_{oo}\, h_{t-1} + b_{oo}) \\
g_t &= tanh(W_{ig}\, x_t + b_{ig} + W_{og}\, h_{t-1} + b_{og}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot tanh(c_t)
\end{aligned}
$$

with initial conditions $c_0 = 0$ and $h_0 = 0$. $\odot$ denotes the Hadamard (element-wise) product. The activation function σ is the sigmoid function and *tanh* is the hyperbolic.

The four internal gating mechanisms regulate the flow of information, enabling the model to retain or discard information over long time horizons. The input gate regulates what to add to i_t, the forget gate f_t what to discard from the cell state, the cell input g_t what to add to the long-term memory and the output gate o_t what to add to the hidden state. This way, the system can learn how information should flow to properly update the current state [7].

LSTMs can be stacked feeding the hidden, latent state, as an input to another LSTM to further introduce model complexity [11]. To make a prediction, the final hidden state is used. In Figure 3.2, this is shown as a linear layer, which maps to the output target p_t. One could instead also pass the output through a FFN to further introduce non-linearity and interpolating ability. The final layer maps the hidden representation to the target space:

$$\hat{y} \in \mathbb{R}^{d_{\text{out}}}.$$

Another example of a RNN is the Gated Recurrent Unit (GRU), which lacks the context vector c_t and the output gate, making it have fewer parameters than the LSTM [12].

3.2.3 Transformer

A more recent practice in sequential learning is the Transformer architecture presented by Vaswani et al. [8]. It was developed primarily for NLP tasks, which are sequential problems (a sentence is a sequence)and is now the foundational building block of modern LLMs. The Transformer develops the concept of attention, a mathematical way of relating information and correlation between past sequences [8]. Unlike the LSTM, all past information is explicitly available at the final step, as the full sequence is processed at

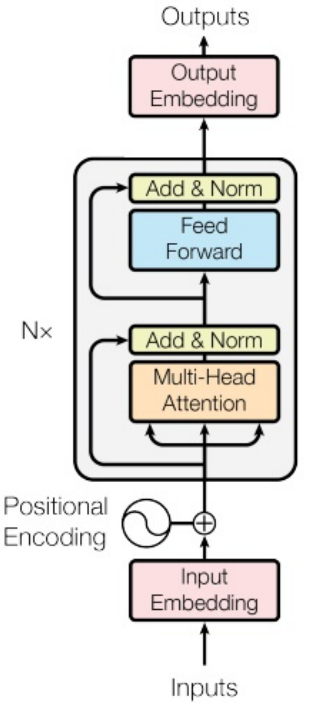


Figure 3.3 *A Transformer encoder block. The input is mapped to a latent space in the model dimension and a positional encoding is added to differentiate temporal dependencies in the system. After that, the sequence is processed in N Transformer blocks, which consist of an attention layer and a FFN. Finally, the output in the targeted space is mapped from the encoded information in the last slice of the sequence. Picture taken from [8].*

once. There is no forget gate or embedded long-term state. By using attention, the Transformer can capture relevant information from the entire sequence, thereby hopefully capturing all relevant spatial–temporal dependencies. The attention function is given by:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

The Transformer function: The first step is that the input sequence is first mapped into a latent space representation. In LLMs this is done by mapping words, or rather tokens, into a vector space representation via a pre-defined mapping. Here this step is omitted, as numerical values are already used. Instead, the input is mapped through a learnable linear transformation so

that the model dimension is assigned to the input sequence at each time step:

$$X_{in} \in \mathbb{R}^{T \times d_{\text{in}}} \longmapsto X_{emb} \in \mathbb{R}^{T \times d_{\text{model}}}$$

where T denotes the length of the input sequence $\{1, \ldots, t\}$, d_{in} is the number of input features and d_{model} represents the internal model (embedding) dimension. To this embedded sequence, X_{emb}, a positional encoding is added, lets revisit that later.

Next, the multi-head attention block is constructed by splitting the model dimension into h attention heads. Consequently, each head operates on a latent subspace with dimension

$$d_k = \frac{d_{\text{model}}}{h}$$

For each attention head and each time step in the sequence, separate linear transformations are applied to the embedded input sequence to generate the queries, keys and values:

$$\begin{aligned} Q_i &= X_{emb} W_i^Q \\ K_i &= X_{emb} W_i^K \\ V_i &= X_{emb} W_i^V \end{aligned}$$

with learnable parameter matrices:

$$W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_k} \quad i \in \{1, \ldots, h\}$$

This operation projects the input into h distinct latent subspaces, yielding the head-specific representations:

$$Q_i, K_i, V_i \in \mathbb{R}^{T \times d_k} \quad i \in \{1, \ldots, h\}$$

The reason for possibly using multi-headed Attention is to allow the model to attend to information from multiple representation subspaces. This also improves efficiency as it enables parallel computation and lower total dimensionality.

Continuing, each head computes attention independently:

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i) = \text{Softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d_k}}\right) V_i$$

The outputs of all heads are concatenated along the feature dimension:

$$\text{Concat}(\text{head}_1, \ldots, \text{head}_h) \in \mathbb{R}^{T \times (hd_k = d_{model})}$$

and projected back to the original embedded space using a final linear transformation:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W_O$$

where $W_O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ is a trainable matrix.

Returning to the attention block (seen in Figure 3.3), one important component has not yet been mentioned: masking. Masking is applied to the QK^T matrix to zero out attention weights that correspond to future information. This prevents information-leakage across the sequence and the model from learning dependencies on data that would not be available at prediction time.

After the Attention block the suggested changes to each vector in the sequence is added to the original input and then normalized [13]:

$$Z = \text{LayerNorm}(X_{emb} + \text{MultiHeadAttention}(X_{emb})) \in \mathbb{R}^{T \times d_{model}}$$

Where *MultiHeadAttention* denotes the mathematical function of the attention block. Then each time-step, column $Z[:]$, is propagated through a FFN (usually two layers) independently and that change is then added to the embedded vector:

$$\hat{Z} = \text{LayerNorm}(Z + \text{FFN}(Z)) \in \mathbb{R}^{T \times d_{model}}$$

updating the newly changed information from the attention. The residual connections allow the model to preserve information from earlier representations while learning incremental refinements, thus helping mitigate vanishing gradient issues in deep Transformer architectures [8].

This constitutes a full Transformer block, as can be seen in Figure 3.3. Often several blocks are used after each other. Once the last block has been applied, the last slice, or token, is taken as the representation of the hidden state and used for prediction. The last vector:

$$h_t = \hat{Z}[t] \in \mathbb{R}^{d_{model}}$$

is finally mapped to the target space:

$$h_t \longmapsto \hat{y} \in \mathbb{R}^{d_{out}}$$

by an affine transformation.

The last thing to address is the positional encoding mentioned in the first step. Because the attention mechanism is permutation-invariant, it lacks an inherent sense of sequential order. By a positional encoding, a small "indexing", the model can incorporate temporal context, allowing the attention mechanism to distinguish and weight information based on its relative or absolute position within the sequence.

The original paper suggested a sinusoidal positional encoding. This is also what is used in this work. Each position p is mapped to a deterministic vector whose components are defined as:

$$\mathrm{PE}(p, 2i) = \sin\left(\frac{p}{10000^{2i/d_{\text{model}}}}\right)$$
$$\mathrm{PE}(p, 2i+1) = \cos\left(\frac{p}{10000^{2i/d_{\text{model}}}}\right),$$

where i indexes the embedding dimensions. Although the positional encodings are absolute in their definition, the sinusoidal structure enables the attention mechanism to recover relative positional information, since a positional shift can be represented as a linear combination of positions.

3.3 State of The Art

Previous work at ESS explored the feasibility of applying ML at ESS. Despite challenges related to data availability, the study proposed a viable workflow for conducting ML projects [14]. As part of this work, initial attempts were made to implement an LSTM model to predict the TMCP return temperature, `"TT-512001"`. These experiments showed that using only a limited set of PVs in a purely time-series prediction setting yielded limited accuracy for reliably modeling or simulating the system. Performance was also constrained by the relatively small amount of available data at the time. However, this was outside the primary scope of the study, which focused on establishing a practical workflow for an ML project rather than on developing a full-scale high-accuracy model.

As discussed before, complex, unknown, unobserved behavior cannot be modeled through white- or grey-box methods, as they build on physical insights into the system [5]. Linear black-box models or linear time-series models, such as the ARMAX family, propose a possibility for modeling systems with unknown dynamics. Linear black-box models share the property

that they can easily be converted into a linear state-space representation, which allows traditional control methods to be utilized for analysis.

However, as soon as the system is non-linear, either in a general model or around certain operating points, these models break down. There is therefore a need for non-linear models, namely the family of Nonlinear Autoregressive with Exogenous Variables (NARX) models. The family of NARX models is defined by a general function based on past values and exogenous variables, which in a control setting possibly includes control input as an exogenous variable [5]. Sequential black-box models for dynamic systems and multi-horizon forecasting generally fall into two categories: autoregressive and sequence-to-sequence architectures [15].

3.3.1 Autoregressive Models

Autoregressive models generate multi-horizon forecasts by iterating a one-step-ahead prediction multiple times. The objective is to learn a one-step state transition function that can be iterated for multi-step purposes:

$$X_{k+n} = f^{(n)}\big(\cdots f^{(1)}(X_{k-m:k}, U_{k-m:k}), U_{k+n-m:k+n}\big)$$

A common autoregressive approach is the use of RNN. These architectures often work well for sequential tasks, but simpler variants suffer from exploding and vanishing gradient problems [9]. As previously introduced, the LSTM (autoregressive) and Transformer (possibly autoregressive) architectures mitigate these problems and their base architectures provide a suitable foundation for sequential tasks and are widely used in sequential learning applications.

DeepAr: Autoregressive RNN models can also be probabilistic, estimating a distribution of paths instead of a single trajectory [16]. This is done by conditioning future values on past values and predictions:

$$p(x_t \mid x_{<t}) = \mathcal{P}\big(\theta_t\big), \quad \theta_t = \mathrm{RNN}(x_{t-1}, h_{t-1}, u_t)$$

where $\mathcal{P}$ is a chosen probability distribution (e.g. Gaussian) and u_t is a covariate or exogenous variable. The specific RNN could be a GRU or a LSTM. During inference, DeepAR recursively feeds sampled predictions back into the model to generate probabilistic forecasts over multiple steps.

3.3.2 Sequence to Sequence Models

The sequence-to-sequence approach learns a direct mapping from an input time window to a sequence of future predictions. This is commonly implemented using encoder–decoder architectures, where an input sequence is

mapped to a latent representation that summarizes the relevant temporal information. A decoding module then uses this representation, together with exogenous inputs both current and future, to generate a forecasted sequence [15]. Such models are often implemented using attention-based architectures designed to enhance long-range dependencies [17]. The model function can be described as:

$$\{X_{k+n}, \ldots, X_{k+1}\} = f(X_{k-m:k}, U_{k-m:k})$$

TFT: The Temporal Fusion Transformer is an architecture designed for interpretable multi-horizon time-series forecasting [15]. It integrates recurrent layers, such as LSTM and GRU cells, to capture local temporal patterns, together with transformer-based attention mechanisms to model long-range temporal and cross-variable dependencies. The model supports the inclusion of known future exogenous inputs and combines elements of iterative temporal modeling with direct multi-horizon sequence-to-sequence prediction

By incorporating future exogenous variables, the model can predict time-series with known future shocks, as well as other co-variables affecting the time-series. This provides a very flexible model in which indicator variables can also be included.

3.3.3 Control Theoretic Contributions

The autoregressive approach closely resembles classical control-theoretic system modeling, where discrete-time system dynamics are identified either from first principles or via data-driven approximation methods. However, other modern approaches exist.

HIPPO: One prominent approach is the High-Order Polynomial Projection Operators (HiPPO) framework, which utilizes a recurrent structure to maintain long-term memory. In this framework, inputs are first mapped into a continuous latent space, after which the long-term hidden state representation, $c(t)$, is continuously updated using structured HiPPO matrices [18].

$$c_{k+1} = Ac_k + Bf_k.$$

The central idea of HiPPO is to represent the entire input history as time-dependent coefficients of an optimal projection onto a fixed basis, defined with respect to a time-varying measure. Concretely, the latent state stores coefficients:

$$c_k^n = \langle f_{\leq k}, g^n \rangle_{\mu_k},$$

where $\{g^n\}_{n=0}^{N-1}$ is a chosen basis and μ_k is a measure. In the HiPPO-LegS (Legendre-Scaled) variant, this basis consists of orthogonal polynomials motivated by classical approximation theory, allowing the projection coefficients to be updated recursively for sequence data:

$$A_{nk} = \begin{cases} \sqrt{(2n+1)(2k+1)}, & n > k, \\ n+1, & n = k, \\ 0, & n < k, \end{cases} \qquad B_n = \sqrt{2n+1}.$$

This way, the HiPPO framework mitigates long-term issues by encoding time-dependent coefficients in a polynomial basis. This representation is then updated at each time step.

S4: Another approach is the use of learned state-space models, in which the state-space matrices are directly regressed from data [19]. In practice, however, such learned state-space models often perform poorly when modeling long-range dependencies. The standard formulation is:

$$\begin{cases} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{cases}$$

By leveraging the HiPPO matrix framework, Structured State Space Sequence Model (S4) is able to effectively track long-range dependencies. In S4, the state matrix is parameterized using the HiPPO construction and subsequently diagonalized to Λ via a unitary matrix V, enabling efficient computation of iterative state updates. In addition, a low-rank correction term PQ^T is introduced so that, together with the input encoding vector B and the output map C, the model consists of a small set of trainable vectors that can learn the system dynamics end-to-end. The resulting parameterization of the state matrix is:

$$A = V\Lambda V^T - PQ^T$$

System class models: In Forgione et al. [20], an in-context learning paradigm for system identification is explored. A single model is trained and validated on several different dynamical systems sampled from the same system class. Rather than learning one model per system, the model learns a full class of systems with a given behavior. Fine-tuning the model later to a specific system then requires less training.

Similar to the data available for the cooling and cryogenic control systems at ESS, data was generated as input-output trajectories. Data was generated in an open-loop by feeding the system random, Gaussian, inputs, hopefully

exciting the system sufficiently such that the model is able to learn the system features through the resulting trajectories.

Two classes of systems were generated in [20], one being a set of LTIs and the other Wiener-Hammerstein systems. For the LTI case, systems were generated in state-space form with randomly sampled dimensions and system matrices. In particular, the state matrix was constructed such that its eigenvalues lie within the unit circle, with magnitude and phase constrained to produce stable and moderately oscillatory dynamics:

$$\mathcal{Z} = \left\{ z \in \mathbb{C} \;\middle|\; |z| \in (0.5, 0.97),\ \arg(z) \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right) \right\}.$$

This ensured that all generated systems shared key qualitative properties while still exhibiting variability in their responses.

For the Wiener–Hammerstein (WH) systems, the data generation process extended the LTI case by introducing nonlinearities. These systems were constructed as a cascade of two linear blocks with a static nonlinear function in between. The linear blocks were generated similarly to the LTI systems. The nonlinear block was implemented as a FFN with randomly initialized parameters. This results in a non-linear, less predictable, class of systems.

Using a large Transformer architecture, an accurate prediction performance across a wide range of generated systems was achieved, demonstrating its potential for modeling. However, this comes at the cost of increased complexity, as a larger model (1–85M parameters) is needed to capture the behavior of an entire class of systems. In contrast, the conventional approach would be to fit a smaller model to each individual system, which renders more computationally efficient models for the given individual system.

3.3.4 Key Takeaways

Continuing with the system-class learning, in general, the primary benefit of meta-modeling (that fits an entire class of systems, or a class of problems) is the reduction of system-specific engineering. From this perspective, its main value lies in identifying a robust architecture and modeling pipeline that performs reliably across different systems. However, this does not necessarily imply that a single trained model should be deployed universally. Instead, the practical utility of meta-modeling may lie more in the transferable modeling framework than in a single class-wide model itself, whose performance may still be constrained by system-specific characteristics and whose size and complexity may need to be significantly greater than those required for an individual problem.

Therefore, for systems at ESS, the most viable approach seems to be fitting a single model per system, while creating a system-wide modeling framework such that it fits most control systems at ESS.

Going back to previous theory, the only type of model that fits the control-system perspective is the autoregressive one. This is due to the fact that the next-step prediction is based on exogenous control input, depending on the current state. Predicting an entire sequence at once is therefore not possible, since control inputs several steps in the future are not known in advance.

As for the traditional approaches versus modern ones, the problems this work faces likely need to use the most recognized state-of-the-art approaches to be able to capture intricate dynamics and model spatio-temporal dependencies well. The modeling will therefore make use of the Transformer and LSTM architectures for autoregressive predictions.

4

Modeling

4.1 Data Collection

Control of the ESS accelerator and its supporting technical systems is based on Experimental Physics and Industrial Control System (EPICS). EPICS is an open-source control framework developed for large-scale scientific facilities and was chosen at ESS to meet the performance demands of the accelerator system that industrial solutions could not provide. Consequently, all control systems, including cooling and cryogenic systems, are implemented using EPICS. In EPICS, Input/Output Controllers (IOCs) interface with hardware and publish data as PVs [21]. These PVs represent measurements, setpoints and system states and are accessed via the Channel Access and PVAccess communication protocols [22, 14].

Operational data in EPICS is archived using the EPICS Archiver Appliance, which stores selected PVs as time-series data. At ESS, it archives approximately 1.1 million PVs, corresponding to about 3 PB of data annually. Archived data can be accessed through tools such as Python APIs (`py-epicsarchiver`), MATLAB, Grafana and CS-Studio's Data Browser, enabling both visualization and programmatic access [21]. CS-Studio provides direct plotting and graphical user interface where system layouts and their associated PVs can be seen, which is used for monitoring and control of the systems. For more sophisticated analysis the Python API `py-epicsarchiver` can be used to access data. EPICS allows for flexibility and IOCs connected to a simulated model, not a physical measurement, can be integrated in the control system, which allows for ML models to be used in the control at ESS [14].

The selection of the relevant PVs was done in collaboration with ESS system experts to ensure all critical plant dynamics were captured.

4.1.1 PV selection for CMS

The main objective of the model is to capture the evolving state of the CMS. The system is manipulated mainly through three controllers. The feed temperature controller acts on control valves "CV-31840" and "CV-31450", which regulate the desired feed temperature "TI-31610". The return temperature controller operates a single valve, "CV-51870", which injects warm helium to stabilize the return temperature to the TMCP, "TI-512001". Then there is the TMCP high-pressure controller, which controls the desired pressure in the TMCP. The high-pressure controller regulates the specified pressure, "PI-217001", through valves "CV-21600" and "CV-21700". There is also a pressure controller acting on the CMS system pressure, "PT-62073", through valve "CV-62029" on the buffer tank in the Coldbox.

Finally, one important controller is the added heater "EH-82650", which warms the system at the target end to simulate a heat load from the actual beam or to provide a more even heat load once the beam is on target. It is important to include this in the model to explain heat disturbances in the data. Once the beam is on target, this must also be included when training the model. For now, the model describes the system in a pre-beam-on-target operating regime.

There are many more control loops acting on the system, but these are not manipulated through changing setpoints. These valves are operated by PIDs to keep the system in a steady state. Since they are not actively operated, the idea is to neglect them as system inputs, effectively including their influence in the model dynamics. Consequently, the model represents the closed-loop behavior of the certain aspects of the system and only valves that receive new setpoints and induce changes in the system state are included as inputs.

Target PVs are measurements of the system state, that is, temperatures, pressures, pressure differences and flows. The obvious ones that need to be included are those that the controllers regulate on. These are "TI-512001", "PI-217001", "PT-62073" and "TI-31610".

Further, there are some measurements that reveal interesting information about the CMS state and can be used to observe how the system evolves. These are therefore included. The obvious ones according to system experts are "TT-628085", "TT-62087", "PT-62073", "TT-61300", "TT-61700" and "PDT-62673".

The borderline case concerns variables that are not particularly interesting as targets but encode information that could help improve model performance.

The challenge is that these variables also become targets, since the model is designed to predict the entire system state. The question is therefore whether more information can be introduced without significantly increasing the difficulty of prediction, or whether this instead adds unnecessary complexity to the prediction task.

Both the LSTM and the Transformer work by encoding information in a latent space. This encoding contains information about what is happening in the system. By adding an extra target dimension, this encoding does not necessary change, but rather the output mapping to the target space. That is, if an additional target dimension is added that is strongly correlated with an existing target, this induces little additional complexity for the model. However, an extra feature that does not correlate with any existing target introduces more complexity for the model to handle. It therefore becomes a trial and error process of finding interesting variables to include for information without adding unnecessary complexity.

Note: If the objective of the modeling task changes – for example, to focus on the control behavior of a single valve – the same modeling approach can be applied using a simpler model. In that case the model may include only one control variable and one target variable to only capture the valve dynamics and its effect on the corresponding PV.

The selected PVs used in the final model can be seen in Table 4.1.

4.1.2 PV selection for the THCS

For the THCS, the selection principle follows the same principle as for the CMS: control variables are chosen among actuators that directly influence operation, while target variables represent temperatures, pressures and flows that characterize the thermal state of the cooling system.

The selected control variables include the main helium compressor speed (`"SC-002"`), the primary cooling water pump speeds (`"SICA-112"`, `"SICA-212"`, `"SICA-312"`) and the feed-water control valve positions (`"TCV-116"`, `"TCV-216"`, `"TCV-316"`). In addition, inlet feed-water temperatures (`"BC-TT-129"`, `"BC-TT-229"`, `"BC-TT-329"`) and differential pressure measurements (`"BC-PDT-118"`, `"BC-PDT-218"`, `"BC-PDT-318"`) are included as auxiliary control variables (or boundary condition, "BC", variables), as they directly affect the system behavior without themselves beeing controlled.

The target variables are selected to capture the thermal behavior of the helium system. In particular, the helium inlet and outlet temperatures of the

Table 4.1 *Control and Target Signals Used in the model for the CMS*

Control Variables	Target Variables
CV-21600 [%]	TT-62117 [K]
CV-21650 [%]	TT-62119 [K]
CV-21700 [%]	PLC-001 [g/s]
CV-21750 [%]	TT-61300 [K]
CV-31450 [%]	TT-61500 [K]
CV-42310 [%]	TT-61700 [K]
CV-51850 [%]	TT-62085 [K]
CV-51860 [%]	TT-62087 [K]
CV-51870 [%]	PI-512001 [bara]
CV-51880 [%]	PI-217001 [bara]
CV-62029 [%]	PT-62073 [bara]
P-62010 [rpm]	PT-62075 [bara]
P-62011 [rpm]	PDT-62067 [Δ kPa]
EH-82650 [W]	TI-31610 [K]
	TI-512001 [K]
	PI-51100 [bara]

water heat exchangers (`"TT-116"`, `"TT-216"`, `"TT-316"`, `"TT-120"`, `"TT-220"`, `"TT-320"`) are included to describe heat transfer from the utility water cooling exchange. Furthermore, the core helium temperature measurements (`"TT-002"`, `"TT-003"`and `"TT-005"`) are selected as primary targets.

To provide additional context on the thermodynamic state of the helium circuit, the helium mass flow rate (`"FT-002"`) and pressure measurements (`"PT-003"`, `"PT-005"`, `"PT-007"`) are also included. Together, these process variables provide sufficient observability of the system while maintaining a compact and physically interpretable variable set suitable for data-driven modeling and control.

The selected PVs used in the final THCS model are shown in Table 4.2.

4.1.3 Sampling Frequency

The raw EPICS data, initially logged at roughly 14 Hz, is resampled to a lower target frequency. For each new sampling interval, the value is computed as the mean of the raw data points within that window.

CMS: Since the dynamics of the CMS are, in some sense, slow, using a sampling interval of 30 s is sufficient to capture the system dynamics. The interesting forecasting horizon is at least a couple of minutes (5–10 min), which is long enough for noticeable changes to occur and for the system to

Table 4.2 *Control and Target Process Variables Used in the THCS Model*

Control Variables	Target Variables
SC-002 [rpm]	TT-002 [K]
SICA-112 [%]	TT-003 [K]
SICA-212 [%]	TT-005 [K]
SICA-312 [%]	TT-116 [K]
TCV-116 [%]	TT-216 [K]
TCV-216 [%]	TT-316 [K]
TCV-316 [%]	TT-120 [K]
BC-TT-129 [K]	TT-220 [K]
BC-TT-229 [K]	TT-320 [K]
BC-TT-329 [K]	TT-154 [K]
BC-PDT-118 [kPa]	FT-002 [kg/s]
BC-PDT-218 [kPa]	PT-003 [kPa]
BC-PDT-318 [kPa]	PT-005 [kPa]
	PT-007 [kPa]

exhibit its time-delayed responses (the time delay between the TMCP and the CMS is approximately 10 minutes).

The entire system evolves slowly, with small but noticeable changes at 30 s intervals. Nevertheless, a naive predictor that forecasts the same value one step ahead (assuming zero change, $X_{k+1} = X_k$) performs reasonably well.

Finding an optimal sampling rate is essentially a trade-off between allowing the model to observe sufficient detail in the signal without it being overly smoothed by averaging and avoiding sampling too quickly, which would make the data noisy and nearly constant over several consecutive steps. The behavior of the buffer tank pressure controller valve `"CV-62029"` can be observed in two examples shown in Figure 4.1. The top panel, that shows 10 min data, shows the valve remaining relatively constant, with a change of approximately one percentage point. In the lower panel, that shows 90 min data, the system exhibits pressure oscillations, likely caused by the vaporization process in the buffer tank and the pressure controller inducing oscillations. Further, inspecting the temperature dynamics of the system, it can be seen in Figure 4.2 that 30 s and 1 min sampling frequency track the dynamics fairly well. A sampling interval of 30 s was selected for modeling the CMS, as it demonstrated strong performance even when evaluated at fixed prediction horizons (e.g. 1 min, 5 min and 10 min). This improvement is likely due to the higher data density and greater signal detail compared to a 1 min sampling interval. At the same time, the 30 s interval remained sufficiently smooth, whereas a 20 s sampling interval led to poorer performance, likely as a result

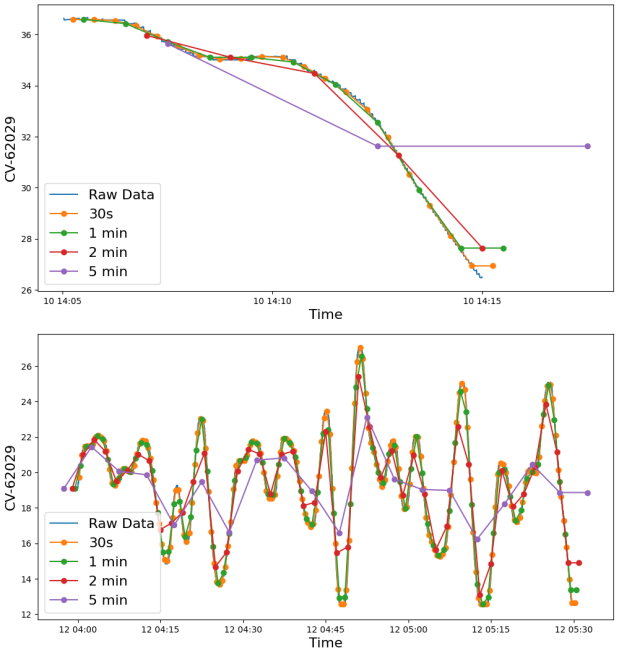


Figure 4.1 *Example plots of the buffer tank valve "CV-62029" position [%] sampled at different intervals. The top picture shows the valve position on 2026-02-10 from 14:05 to 14:15 and the lower picture shows the valve position on 2026-02-12 from 04:00 to 05:30.*

of increased noise.

THCS: The THCS showed similar time dynamics to the CMS, likely because it is also a cooling system relying on similar valves, control and heat exchangers. This made 30 s a suitable sampling choice for that system as well.

4.1.4 Data Availability

CMS: The data available for the CMS consisted of five cooldowns from the past year. Previous to that the system had not been run live with liquid hydrogen. This operations consequently corresponded to five cooldowns, five warmups and rather steady-state operation in between with little excitations and varying setpoints. As for the cooldowns and warmups those were also operated within a strict protocol, not rendering much variation in operation.

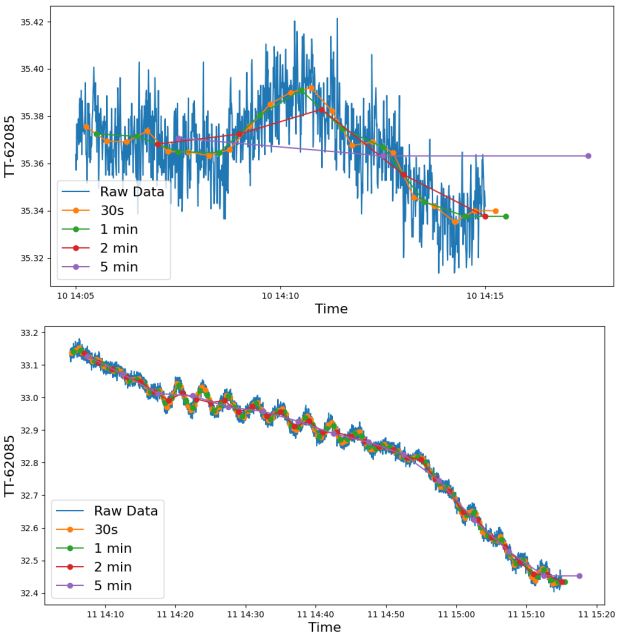


Figure 4.2 *Example plots of the CMS temperature [K], namely element* `"TT-62085"`*, sampled at different intervals. The top picture shows the temperature over a shorter interval, from 2026-02-10 14:05 to 14:15, corresponding to the same interval as the left valve plot. The lower picture shows a longer interval, from 2026-02-11 14:05 to 15:15.*

Because the data were collected in a pre-beam-on-target regime, the available dataset was restricted to a narrow operating range that lacked sufficient system excitation. The total amount of data available was 96 days or 2300 hours. With much of it being steady state operation as can be seen in Figure 4.3. The number of 30s points were 277k.

THCS: For the THCS data was collected whenever the system was in operation, defined by when the compressor `"SC-002"` was active. This generated 84 clusters seen in Figure 4.4. This corresponded to 51 days of operation or 1200 hours, rendering 148k data points. In contrast to the CMS data the THCS operation varied a lot more rendering significantly more informative data. Similar limitations regarding the amount of data and restricted operational regimes existed for the THCS as well.

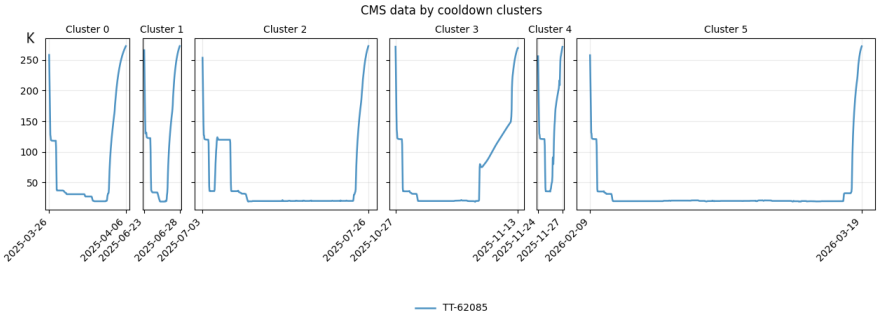


Figure 4.3 *CMS cooldown cluster. As can, somewhat, be seen, the system is operated in steady state for the majority of the time. Even during actual cooling down and warming up the system, the operation is rather slow and not really exciting the system rendering information. One of the CMS temperatures is shown (`TT-62085`).*

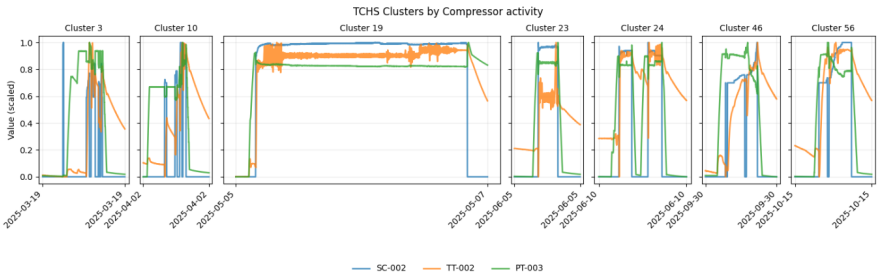


Figure 4.4 *Compressor activity (`"SC-002"`), a helium-side temperature element (`"TT-002"`) and a pressure measurement (`"PT-003"`) plotted for selected clusters of THCS data. All PVs plotted scaled. Each of the 84 clusters has varying operation and contributes with informative data. A few selected PVs are being show. The main goal is not to show any dynamics in detail, but to demonstrate that the data consists of operational clusters (84 in total) with some variability in all of them.*

4.2 Model Selection

The objective is to build a simulator-like model in which control inputs actively control the system. Both the CMS and the THCS operate in closed-loop control. Consequently, any model must also be formulated as a closed-loop model to enable predictions beyond a single time step. While models such as LSTMs and Transformers can make multi-step predictions, this capability is not in its own useful in this context. In a closed-loop system, each predicted step generates a new control input, which in turn affects system behavior. As a result, a sequence-to-sequence approach is not suitable, since future states depend on control actions that are not known in advance. Instead, the model must be autoregressive, allowing it to be integrated into a closed-loop

framework together with the controllers, as illustrated in Figure 4.5 and seen in Algorithm 1. The scheme can be formulated mathematically with both a predictive model f, representing the system dynamics and a control law, C, controlling the system:

$$\begin{cases} X_{k+1} &= f(X_{k-m:k}, U_{k-m:k}) \\ U_k &= C(X_{k-m:k}, U_{k-m:k-1}, r_k) \end{cases}$$

The variables X_k represent the system target variables, i.e. the sensor measurements that the model is trained to predict. These are the endogenous system variables, meaning they are part of the system dynamics and evolve in response to both past system states and applied inputs. In this work, the term target variables refers specifically to variables for which the model predicts the one-step state evolution.

The control variables U_k are the control inputs or the exogenous variables. These typically include actuator signals such as valve positions, pump speeds, or applied power. In the case of the THCS model, U_k, also includes certain sensor measurements that influence the system but are not influenced by it and therefore act as exogenous inputs. An example in the THCS is the conventional water supply temperature, which affects the cooling performance of the system without really being affected by the system itself.

Since persistence is particularly strong and accurate, the models are trained with a skip connection from the previous state X_k, letting the model predict the one-step change. That is, f essentially looks like:

$$f(X_{k-m:k}, U_{k-m:k}) = X_k + \tilde{f}(X_{k-m:k}, U_{k-m:k})$$

Algorithm 1 Autoregressive Control Loop (Deployment)

Require: Initial history window W, Reference signal trajectory $\mathbf{R}_{1:n}$, Steps n
for $k = 1$ to n **do**
1. **Predict State:** $\hat{x}_k \leftarrow f(W)$
2. **Compute Control:** $\hat{u}_k \leftarrow C(W, \hat{x}_k, r_k)$
3. **Update History:** $W \leftarrow \text{Shift}(W, [\hat{u}_k, \hat{x}_k])$

end for
return Predicted trajectory $\hat{\mathbf{X}}$

4.2.1 The Iterative Training Strategy (ITS)

Since the model is required to operate over multiple time steps through iterative predictions, it must be able to capture long-term behavior. Training only

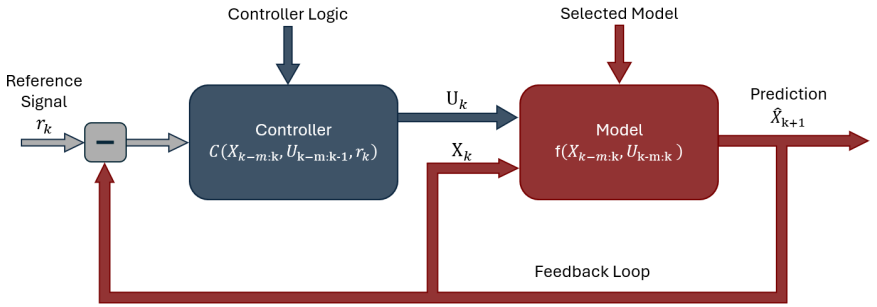


Figure 4.5 *Closed-loop control scheme showing the full intended simulator. This separates control inputs (exogenous variables) and target PVs (endogenous variables) for the model block and shows where and how to apply the desired control law. For training and in this project, U_k is taken from data and only the model block part of the scheme is used.*

a one-step predictor may cause the model to lose its ability to perform autoregressively over multiple steps, failing to capture longer-term dependencies and overfitting on the first-step predictions. By training it to make several multi-horizon predictions, the hope is that the model will be regularized to avoid overly aggressive, overfitted first-step predictions.

An **Iterative Training Strategy (ITS)** is therefore proposed in Algorithm 2 & 3. By training the model on the loss accumulated over several predictions, the goal is for the model to learn realistic system trajectories that remain aligned with the underlying system behavior.

Algorithm 2 Autoregressive Predictions (Training with Data)

Require: Initial history window W, True future controls $\mathbf{U}_{1:n}$, Steps n
for $k = 1$ to n **do**
1. **Predict State:** $\hat{x}_k \leftarrow f_\theta(W)$
2. **Inject Control:** $u_k \leftarrow \mathbf{U}_k$
3. **Update History:** $W \leftarrow \text{Shift}(W, [u_k, \hat{x}_k])$

end for
return Predicted trajectory $\hat{\mathbf{X}}$

As can be seen in Algorithm 2, during training the model makes a single-step prediction. This prediction is then fed back and appended to the input sequence together with the actual control data for the following step. The procedure is repeated for several steps, generating a full predicted trajectory.

Deploying Algorithm 3 during training, the generated trajectory is compared to the corresponding true sequence and the loss is calculated by comparing

Algorithm 3 Iterative Training Strategy

Require: Dataset $\mathcal{D}$, Predictive model f_θ, Horizon n, Learning rate η
for Batch $(W, \mathbf{Y}_{future}) \in \mathcal{D}$ **do**
1. **Extract Targets:** Separate the future control data and true state trajectories:
$\mathbf{U}_{1:n} \leftarrow$ Controls from $\mathbf{Y}_{future}$
$\mathbf{X}_{1:n} \leftarrow$ Targets from $\mathbf{Y}_{future}$
2. **Autoregressive Predictions with Data:**
$\hat{\mathbf{X}} \leftarrow$ **AutoregressivePredictions**$(W, \mathbf{U}_{1:n}, n)$ ▷ *Executes Algorithm 2*
3. **Trajectory Loss Calculation:**
$\mathcal{L} \leftarrow \text{Loss}(\hat{\mathbf{X}}, \mathbf{X}_{1:n})$
4. **Backpropagation** Compute gradients over the entire computational graph [23]:
$g \leftarrow \nabla_\theta \mathcal{L}$
5. **Optimizer Step:**
$\theta \leftarrow \text{Update}(\theta, g, \eta)$
end for

the two sequences. This way, the model loss is quantified by its ability to follow the true system trajectories induced by the control signals in the data. Note that future control inputs are not given to the model in advance; instead, the model is trained to follow the same control inputs that rendered the true trajectory.

Since the prediction $\hat{x}_{k-1}$ affects $\hat{x}_k$, a derivative with respect to the model parameters θ will split according to the chain rule:

$$\frac{\partial \hat{x}_k}{\partial \theta} = \frac{\partial \hat{x}_k}{\partial \hat{x}_{k-1}} \frac{\partial \hat{x}_{k-1}}{\partial \theta} + \frac{\partial \hat{x}_k}{\partial \theta}$$

In the ITS, the loss, $\mathcal{L}$, is therefore backpropagated through the entire prediction horizon, meaning that, when predicting n steps ahead, the gradients flow through all n sequential prediction steps. Losses from all horizons are backpropagated according to:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{k=1}^{n} \sum_{j=1}^{k} \frac{\partial \mathcal{L}_k}{\partial \hat{x}_k} \underbrace{(\cdots)}_{\text{Chain of derivitives}} \frac{\partial \hat{x}_j}{\partial \theta}$$

$$\implies \frac{\partial \mathcal{L}}{\partial \theta} = \sum_{k=1}^{n} \sum_{j=1}^{k} \frac{\partial \mathcal{L}_k}{\partial \hat{x}_k} \left(\prod_{i=j+1}^{k} \frac{\partial \hat{x}_i}{\partial \hat{x}_{i-1}} \right) \frac{\partial \hat{x}_j}{\partial \theta}$$

By doing so, the model can hopefully learn to make modest one-step predictions such that errors will not cascade through the sequence, hopefully improving long-term performance.

4.3 Training and Criterion of Fit

Criterion of fit: Since the model is fundamentally a one-step predictor, one-step MSE remains an important metric. However, since the objective is strong iterative performance, accuracy over longer prediction sequences are crucial. Consequently, the MSE computed for several different, longer, horizons serves as the main criterion of fit.

A common pitfall in machine learning is to focus blindly on the MSE or a similar measure as the performance metric. In the case of a system like this, with strong persistence and almost Brownian noise-like behavior, a low MSE may appear to indicate a good fit. However, the model might simply be making the naive prediction.

$$X_{k+1} = X_k$$

A necessary metric for evaluating the performance of the model for this system is therefore whether it can outperform the naive predictor. For this, a relative metric is introduced [24]:

$$\textbf{Skill Score} = 1 - \frac{MSE_{model}}{MSE_{naive}}$$

This metric evaluates the model's loss against a naive baseline, meaning performance is always relative to a constant prediction. Scores above 0 indicate an improvement over this baseline, with values above 0 reflecting increasingly better accuracy; for example, scores of 0.5, 0.9 and 0.99 correspond to MSE losses that are 2, 10 and 100 times lower than those of the naive prediction. Because it is a relative measure, a low MSE may not guarantee a high Skill Score if the underlying system change is minimal. Instead, the metric primarily highlights the model's predictive capability during changes in the system. Although the models are trained using MSE, the Skill Score offers a more interpretable measure of performance, since a low loss alone is highly scale-dependent and may say very little about actual predictive performance.

Note: The MSE for each target variable is computed using standardized scaling to $[0, 1]$, preventing variables with larger magnitudes from dominating the average Skill Score across all targets.

4.3.1 Train–Test Split

CMS: When data were collected in the CMS, the cooldown periods split the dataset into five clusters, corresponding to five individual cooldowns. This provided a suggestion for an intuitive way to partition the data, where four cooldowns could be used for training and the remaining one reserved for testing.

However, as a large fraction of the data within each cooldown corresponds to steady-state operation with low variation. Such data are of limited interest when training models intended to capture system dynamics and transient behavior. To address this, an alternative clustering strategy could be used, focusing only on high-variance segments of the data. Specifically, the variance of selected PVs was evaluated over a fixed look-back window. If the variance exceeded a predefined threshold, the corresponding sequence (including a small amount of padding before and after) was retained.

By varying the variance threshold, it was possible to control the amount of retained data. This resulted in training datasets ranging from the full cooldown dataset (approximately 277 k data points) down to the smallest tested subset of roughly 120 k data points. This produced a set of clusters (84 for the 120k dataset).

Having formed clusters enabled an alternative train-test split, where data could be divided randomly according to the clusters. This raised the question of how generalization was actually being tested. Leaving out an entire cooldown for evaluation, tests generalization to a completely unseen operating period. This can be overly restrictive as different cooldowns may be operated under substantially different conditions or regimes. In such cases, the test data may contain too little overlap in operating behavior with the training data.

Conversely, splitting the data based on the formed high-variance clusters allows the model to be exposed to all observed operating regimes during training, while still testing its ability to generalize to unseen high-variance sequences. This split represents a compromise between testing meaningful generalization and avoiding an unrealistically difficult test scenario.

This led to the question of how to split the data, given the desire to use all of it. To enable the cluster based splitting, the data could be split into for example, 150 generated clusters by randomly splitting the data additionally. Such an approach, like the natural clusters, ensures that no data points overlap and can be used if all data is needed for training, even the stale periods.

It remains unclear how to define a reasonable notion of out-of-domain operation when constructing test datasets. For example, leaving out an entire cooldown may constitute an overly strict test, while randomly selected clusters drawn from similar operating regimes may still be effectively in-domain. This therefore remains an open question; however, in practice, splitting the data into randomly selected clusters appeared to introduce a degree of out-of-domain behavior.

Subsequently, the choice was made to train the models of the CMS by splitting the data into 150 randomly divided clusters.

K-fold Cross-Validation: When the available amount of data is limited, it can be difficult to construct a sufficiently large test dataset that can be split into homogeneous subsets such that model performance is evaluated reliably regardless of which specific data are selected for testing. In such situations, k-fold cross-validation can be used to obtain a more robust estimate of model performance [25]. The method works by partitioning the dataset into k subsets, where k-1 subsets are used for training and the remaining subset is used for testing [25]. This procedure is repeated k times, each time leaving out a different subset for evaluation. In this way, the amount of data used for training is maximized, while still ensuring a statistically more rigorous evaluation, as all data points are included in testing once [25].

Due to the limited availability of data for the CMS, particularly data exhibiting meaningful variability, k-fold cross-validation is used to evaluate model performance on the CMS.

THCS: When collecting data for the THCS, operating status was defined as whenever the compressor (`"SC-002"`) was running. This formed 84 clusters (148k data points) from which k-fold Cross-Validation could be performed.

Artificial Systems: More on these later (in section 4.4.1), but for the sake of structuring the report, it is worth noting that 200 clusters were generated out of 500k data points in total. There was no need for a k-fold evaluation here; a simple $85\% - 15\%$ train-test split for each generated system was sufficient.

4.3.2 Implementation

CMS and THCS: After splitting the datasets, input sequences were constructed using a 15-minute history as model input, corresponding to 30 samples at a sampling interval of 30 s. This window length was sufficient for all relevant time-delayed effects to manifest, enabling the models to capture the necessary system dynamics. During iterative prediction, the most recent

prediction was appended to the input sequence while the oldest sample was removed, maintaining a constant input length of 30. All data were standardized using min-max scaling to the range $[0, 1]$, ensuring numerically stable training and smooth gradients.

Training was initialized using only one-step predictions, which is computationally efficient, as it requires only a single forward pass per data point. This enabled more gradient updates per unit time and faster initial convergence. After a number of single-step epochs (120 for the k-fold setting) models reverted back to the best checkpoint and were then introduced to the ITS. For this the prediction horizon varied across epochs. This allowed performance to be evaluated and kept track off over multiple horizon, while favoring accuracy on short horizons which is appropriate for a one-step predictor. In the k-fold setup training was continued for 40 additional epochs doing this.

All models were implemented and trained using the PyTorch framework, which provides efficient GPU acceleration and automatic differentiation. Dropout (of 0.1 in most cases) was used for regularization. Optimization was performed using the AdamW algorithm, combining adaptive gradient updates with decoupled weight decay for improved regularization [26]. Initially training started with a learning rate of $5e-4$ for 120 epochs and after when switching to the ITS, the learning rate was set to $5e-5$ to make learning more stable with the accumulated gradients of the ITS. For the single-step only training the learning rate was also lowered after the 120 first epochs, also running 160 epochs in total to ensure similar conditions for a comparison between the iterative and single-step training.

As for hyperparameter the Transformers tested the following architecture combinations, where black can be seen as the standard and the varying parameters for each combination with a different color. The default, black combination, is called Transformer-128-4-256-2-0.1 later in the text (see section 3.2.3 for information on the Transformer).

$$
\begin{aligned}
d_{\text{model}} &\in \{32, 128, 256\} \\
d_{\text{FFN}} &\in \{64, 256, 512\} \\
n_{\text{heads}} &= 4 \\
n_{\text{layers}} &\in \{2, 3\} \\
\text{dropout} &= 0.1 \\
\text{output mapping} &\in \{\text{Linear map}, \text{2-layer FFN } (d_{\text{latent}} = 64)\} \\
\#\,\text{parameters} &\in \{19\,\text{k}, 271\,\text{k}, 278\,\text{k}, 1\,066\,\text{k}\}
\end{aligned}
$$

For the LSTM the following were tested, the black combination being denoted LSTM-128-2-0.1-FFN64 later in the text (see section 3.2.2 for information on the LSTM):

$$\begin{aligned} d_{\text{model}} &= 128 \\ n_{\text{cells}} &\in \{1, 2\} \\ \text{dropout} &= 0.1 \\ \text{output mapping} &\in \{\text{2-layer FFN } (d_{\text{latent}} = 64), \text{Linear map}\} \\ \#\,\text{parameters} &\in \{84\,\text{k}, 223\,\text{k}\} \end{aligned}$$

A proper hyperparameter optimization study was also tried, but did not yield any significant insights in reasonable runtime, since large models took time to converge.

4.4 Validation

Focusing on the validation aspects; model performance was evaluated using the one-step loss and the iterative loss at fixed absolute horizons:

$$MSE_{30s},\ MSE_{1\,min},\ MSE_{5\,min},\ MSE_{10\,min},\ MSE_{15\,min}.$$

From these metrics, the corresponding Skill Scores were computed:

$$Skill_{30s},\ Skill_{1\,min},\ Skill_{5\,min},\ Skill_{10\,min},\ Skill_{15\,min}.$$

This formulation enables evaluation of iterative prediction performance independently of the sampling frequency, using the naive predictor as a baseline. Model performance was then assessed based on the average skill across all prediction horizons.

For the k-fold validation the Skill Score was simply calculated as the average MSE (using standardized targets) over all k-folds:

$$\textbf{Skill Score}_{k-fold} = 1 - \frac{mean_k(MSE_{model})}{mean_k(MSE_{naive})}$$

4.4.1 Generating Artificial Systems

To validate the modeling pipeline, it is tested on several artificial systems generated from known dynamics. The objective is to assess whether the method can learn systems of varying complexity that share key properties. In order

to create systems similar to the control systems at ESS, several specific design choices must be made. At the same time, to ensure sufficient variability and to generate a diverse set of artificial systems, as many parameters as possible are randomly sampled and differ from system to system. Despite this randomness, the imposed design choices introduce a consistent structure and are selected such that the generated systems resemble real control systems at ESS.

Beginning with the choices; a standard class of systems mentioned previously are the discrete-time LTI systems. For a LTI system the eigenvalues of the system matrix A can be explicitly placed to enforce desired dynamical behavior. Another key design objective is to mimic closed-loop PID-controlled systems, where each operated state is controlled by a single PID controller. Specifically, a system with n total states, m operated (controlled) inputs and o observed outputs can be formulated as:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_{k-d} + w_k \\ y_k &= Cx_k \end{aligned}$$

Where $d \in \mathbb{N}^m$ represents state-specific input delays. To reflect realistic partially observed systems, the total state dimension can be generated dynamically as $n = o + \Delta_n$, where $\Delta_n \sim \text{Geometric}(0.04)$, resulting in an expected value of 25 additional hidden states.

The open-loop system matrix $A \in \mathbb{R}^{n \times n}$ may be constructed to exhibit slowly decaying, stable dynamics characteristic of cooling systems at ESS. A prescribed fraction of the eigenvalues is chosen to be complex-conjugate pairs, with the ratio of real to complex eigenvalues drawn uniformly from $\mathcal{U}(0, 1)$. The eigenvalues are then randomly sampled within a stable region of the positive real complex plane:

$$\mathcal{Z} = \{z \in \mathbb{C} \mid \text{Re}(z) \in (r_{\min}, r_{\max}),\ |z| < r_{\max}\}$$

The upper bound is set to $r_{\max} = 0.8$ to ensure stability, while the lower real bound $r_{\min}$ is drawn uniformly from $(0, r_{\max} - 0.1)$. A real block-diagonal matrix Λ is formed from these eigenvalues using the Jordan normal form [27]. To distribute the modes across all state coordinates, the system matrix is generated via an orthonormal similarity transformation,

$$A = Q\Lambda Q^\top,$$

where Q is obtained from the QR decomposition of a standard normal random matrix [28].

Unlike idealized decoupled systems, the input matrix $B \in \mathbb{R}^{n\times m}$ and output matrix $C \in \mathbb{R}^{o\times n}$ are constructed from baseline identity mappings with added cross-coupling. Specifically,

$$B = \begin{bmatrix} I_m \\ 0 \end{bmatrix} + \mathcal{N}(0, \sigma^2_{\text{coupling}})$$
$$C = \begin{bmatrix} I_o & 0 \end{bmatrix} + \mathcal{N}(0, \sigma^2_{\text{coupling}})$$

where the coupling standard deviation is sampled as $\sigma_{\text{coupling}} \sim \mathcal{U}(0.05, 0.15)$. This construction introduces varying degrees of inter-channel interference while preserving overall controllability and observability.

The system is operated in closed loop using m independent PID controllers. To ensure diversity among the generated systems, the controller gains are sampled independently for each channel from predefined operating ranges,

$$\begin{aligned} (K_p)_i &\sim \mathcal{U}(0.001, 0.1) \\ (K_i)_i &\sim \mathcal{U}(0.0005, 0.01) \qquad i \in \{1, \dots, m\} \\ (K_d)_i &\sim \mathcal{U}(0.0001, 0.001) \end{aligned}$$

To further increase realism, heterogeneous input delays can be applied to the operated channels in order to mimic delays observed in real systems. For each channel, the delay d_i is drawn uniformly from $\{0, \dots, d_{\max}\}$, with $d_{\max}$ beeing 20 steps. To reflect scenarios where only a subset of actuators is affected by delays, 40% of the channels are explicitly forced to have $d_i = 0$. Non-delayed control signals are logged in real time, while their effect on the system state occurs after the corresponding delay, consistent with real system behavior.

To induce operating-point changes and evaluate tracking performance, the reference signal $r_k \in \mathbb{R}^m$ is subjected to step changes drawn from a discrete uniform distribution,

$$r_k^i \in \{-10, \dots, 10\}$$

These step changes occur independently for each channel, with wait times between changes drawn uniformly from $[T_{\min}, T_{\max}]$. The bounds themselves are randomized per system, with $T_{\min} \sim \mathcal{U}(100, 500)$ and $T_{\max} \sim \mathcal{U}(800, 1500)$.

Finally, the process disturbance $w_k \in \mathbb{R}^n$ can modeled as a low-rank load disturbance combined with additive state-specific noise,

$$\begin{aligned} w_k &= Lz_k + \epsilon_k, \\ \epsilon_k &= \mathcal{N}(0, \sigma^2_{\text{noise}}) \\ z_k &\sim \mathcal{N}(0, \sigma^2_{\text{load}}) \end{aligned}$$

The dimension of the latent load vector is scaled as $r = \max(3, n/10)$. The random mapping matrix $L \in \mathbb{R}^{n \times r}$ is subject to a 50% row-sparsity constraint, ensuring that latent disturbances affect only subsets of the system states at any given time. Rows are subsequently normalized. Each component of the latent load z_k evolves as a piecewise-constant process. Analogous to the reference signal, it updates at random intervals, with an update frequency scaled to twice that of the reference step changes.

4.4.2 Generated Systems and Validation Models

All artificial systems were generated with 15 operated states and 25 observed states. Each system was simulated for 500 000 steps and segmented into 200 continuous clusters to enable a cluster-based train–test split, with 15% of the clusters reserved for testing.

For each experimental setting, 10 independent system instances were generated using different random seeds. The only parameters varied explicitly across these systems were the disturbance magnitudes. Four configurations were considered, differing in the standard deviations of the latent load disturbance and the additive noise,

$$(\sigma_{\text{load}}, \sigma_{\text{noise}}) \in \{(0,0),\ (0.1, 0.01),\ (0.5, 0.05),\ (1, 0.1)\}.$$

The different noise levels correspond, given the size of the reference tracking objective, to: no noise, moderate noise, high noise and very high noise. In Figure 4.6 one of the systems can be seen simulated with moderate and high noise levels.

All 10 systems were modeled for each noise level using the same neural network architecture and training procedure. A transformer model was used throughout, with a model dimension of 128, four attention heads, two encoder layers and a feed-forward dimension of 256. The input sequence length was set to 20 time steps. Training was performed for 150 epochs using the AdamW optimizer with a learning rate of $5e-4$ for the single step training and $5e-5$ for the ITS (to make learning more stable with accumulated gradients) and and a batch size of 64.

Model performance was evaluated at prediction horizons of 1, 2, 10, 20and 30 steps similar to the real systems. For each horizon, prediction errors were compared against a naive persistence baseline and skill scores were computed on the held-out test clusters.

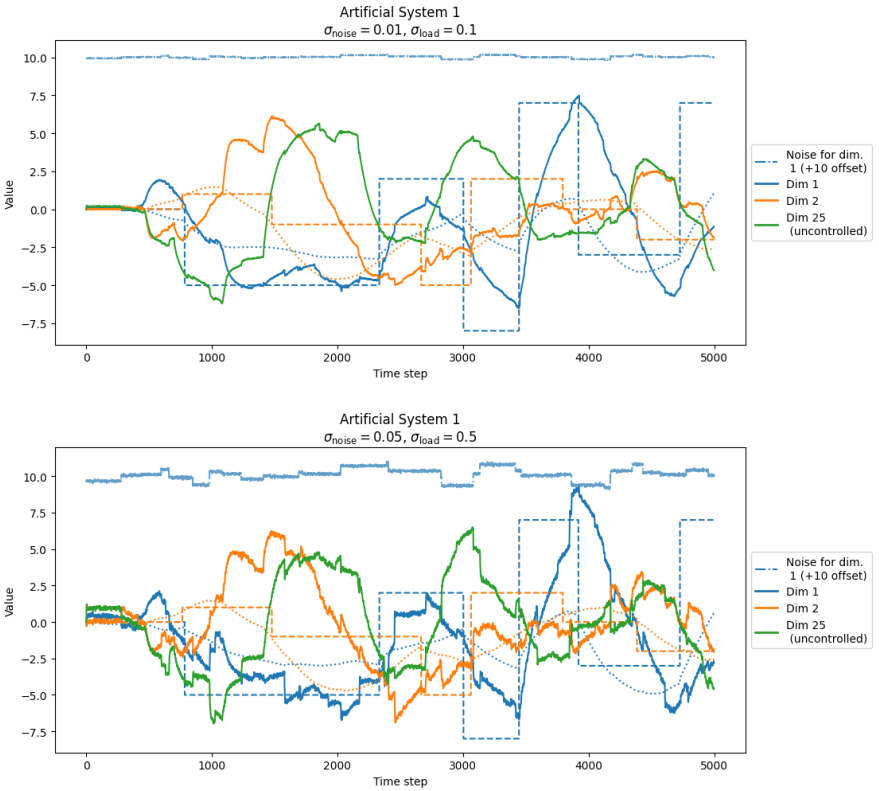


Figure 4.6 *An example of what the simulated data from a specific realization of Artificial System 1 under different noise levels looks like. The reference signal (r dotted), the control signal (u dashed), the output target signal (y line) and the noise (ω split line +10 offset) can be seen for a selected few dimensions. Dim 1 and 2 beeing controlled dimensions and dim 25 an observed one.*

All systems shared $n_{inputs} = 15$ *and* $n_{outputs} = 25$. *The system 1 realization yielded a discrete-time LTI system with* $n_{states} = 42$ *and disturbance rank* 4. *It got moderate coupling (Std* $= 0.1046$*), an eigenvalue real-part range* $[0.1889, 0.7982]$ *and* 83.5% *complex eigenvalues. Per-dimension delays were:* [5, 4, 15, 16, 0, 0, 10, 20, 0, 0, 5, 0, 0, 16, 0].

5

Results and Discussion

5.1 CMS

In Table 5.1, the skill scores based on the mean MSE across all k folds are shown for all tested model types (with varying hyperparameters) and training strategies when evaluated on the CMS data. The results indicate that larger models, with a latent dimensionality of 128, performs better than the smaller ones, but that further increasing model size did not result in better performance. It is likely that a larger model is able to easier find a good solution that captures the complex dynamics, but that a too large of a model simply is too data-demanding for good performance. Model size seems to be the deciding factor for performance with the LSTM and Transformer architecture posting similar numbers given their parameter count.

The LSTM model was able to find better solutions for shorter prediction horizons and the Transformer more tracks system behavior more accurately across longer horizons. This may possibly be due to the Transformer's ability to capture long-range dependencies via attention mechanisms, whereas the LSTM represents such dependencies only through its latent state.

As for the proposed ITS, it appears to perform better, especially for longer horizons, as intended.

CMS specific performance: As for the performance on the actual CMS models seem to beat the naive predictions on average over the entire dataset. What is interesting, however, is that performance varied a lot depending on the fold, with some performing significantly worse not beating the naive prediction. This indicates that there are data segments where the models are not able to generalize on the CMS. This is also what drags down the average skill scores to pretty low ranges. The reason for the uneven performance could be the lack of informative data and the restricted operation of the

CMS, some folds might happen to include testing data that is from a region of operation unseen in training, therefore making it hard for the models to generalize on them.

This can also be seen when using the models to plot sequences (see Section "Simulated Sequences" in the Appendix, Figures 7.1 and 7.2). With a skill score of $\lesssim 0.5$ on short ranges and $\lesssim 0.25$ on longer ranges, many trajectories deviate from the actual data when simulating with the iterative strategy. This divergence indicates that the model either has not learned the dynamics well enough, or that disturbances or unmodeled dynamics affected the paths in the data; the latter would mean the model did in fact make a "true" prediction. However, this seems unlikely, as simulated sequences for low-scoring folds of the CMS often diverged and failed to accurately track the test trajectories. The remaining explanation is that performance is limited by the available data, which lacks sufficient informative, system-exciting examples.

To conclude, the obtained CMS models are not accurate enough for reliable simulation right now. This is most likely due to the limited amount of informative, system-exciting data rather than intrinsic complexity that the proposed model classes cannot capture. To know fully the model potential more informative data has to be waited for. This data will soon be available as ESS will start up its beam-on-target operation, significantly increasing the amount of system-exciting, varying, data.

5.2 THCS

In Table 5.2, the model performance on the THCS can be seen. Both the LSTM and the Transformer perform very well, especially when using the ITS. However, the LSTM achieves the best overall performance, appearing to find a better generalizing solution given the limited amount of data. Since the THCS has fewer long-range temporal dependencies, the advantages of using a Transformer architecture may not be fully needed in this case.

As for the ITS performance looks better across all models for longer horizons. The speculation was that it could force models to be "careful" with its first prediction to accommodate longer trajectories, thus finding smoother, more regular solutions. However, there is no way, through these results, to know what underlying mechanisms are behind this. Still the result show that the idea could prove useful for training autoregressive models on multi-step prediction tasks.

THCS specific performance: As for the specific THCS performance; the results were significantly better than for the CMS. A likely reason for this is that the data were more informative, as the number of signals (PVs) used was roughly the same and the complexity of the systems were similar. The CMS may have a slightly more complex nature due to its cryogenic properties. There is obviously no way of knowing which factor is the deciding one without access to more data for the CMS (testing the fundamental limit of the modeling approach), but the overall impression is that data quality is the deciding factor

The models exhibited good performance across all horizons with a skill score of ≈ 0.8 for longer horizons. This is something that is supported when evaluating simulated sequences compared to actual data (see section "Simulated Sequences" in the Appendix). As can be seen in Figure 7.3, sequences stay true to observed behavior, which was typical when simulating several sequences. Therefore, the THCS model could potentially be used. However, because its out-of-domain generalization capabilities could not be properly evaluated, one must proceed with caution.

5.3 Artificial Systems

In Table 5.3 and 5.4 the performance of both a Transformer and a LSTM model on artificial systems with varying noise levels can be seen. The models were trained using the iterative strategy. The results show that, even for general systems with unobserved dynamics, significant delays and substantial noise, the models are able to learn the underlying dynamics and its iterative predictions perform significantly better than those of the naive baseline. Comparing the Transformer and the LSTM architectures it can be seen that the LSTM perform better on the artificial systems.

Looking at the min-max deviations it shows that some systems, due to their nature, are harder for the models to learn. Causes for this could possibly be less stable, more oscillatory systems, higher coupling between states and higher number of unobserved states interfering. This verifies, to some extent, that the randomized generation succeeded in creating different systems with varying complexity, resulting in the validating being done on a variety of systems as was intended.

Naturally, higher noise levels in the system lead to less accurate predictions. There are two possible reasons for this. First, increased noise makes the underlying noiseless system dynamics more difficult to learn and also increases the risk of overfitting, which can degrade model performance. Second, the

presence of noise in the test data causes trajectories to deviate from the true system dynamics. As a result, the model may produce an accurate prediction of the underlying dynamics, but the observed data are perturbed by noise and therefore differ from the predicted values. This discrepancy is then counted as a prediction error, even though the model has captured the correct underlying behavior.

Overall, validation on the artificial systems indicates that the modeling approach is promising when sufficient data are available. Despite the generated systems being high-dimensional, complex and containing unobserved dynamics, the models are able to capture these characteristics and produce predictions that are significantly better than those of the naive predictor, particularly at reasonable noise levels.

Models and Average Test Skill Scores

Horizon (minutes)	**Transformer** 32-4-64-2-0.1 19k params ITS	**Transformer** 128-4-256-2-0.1 271k params ITS	**LSTM** 128-2-0.1-FFN64 223k params ITS
0.5	0.3816	0.3879	0.4167
1	0.4388	0.4476	0.4915
5	0.2079	0.2927	**0.2968**
10	0.1021	**0.2284**	0.2032
15	0.0301	**0.1824**	0.1330

Horizon (minutes)	**Transformer** 32-4-128-2-0.1 19k params Single step training	**Transformer** 128-4-256-2-0.1 271k params Single step training	**LSTM** 128-2-0.1-FFN64 223k params Single step training
0.5	0.3956	0.4130	**0.4340**
1	0.4423	0.4673	**0.4972**
5	0.1962	0.2429	0.2581
10	0.0852	0.1425	0.1285
15	0.0157	0.0683	0.0222

Horizon (minutes)	**Transformer** 256-4-512-2-0.1 1,066k params ITS	**Transformer** 128-4-256-2-0.1-FFN64 278k params ITS	**LSTM (Single cell)** 128-1-0.1-Linear Map 84k params ITS
0.5	0.3674	0.3716	0.3823
1	0.4210	0.4213	0.4133
5	0.2370	0.1832	-0.1741
10	0.1522	0.0567	-0.6977
15	0.0795	-0.0309	-1.3242

Table 5.1 *Average CMS test skill scores across k = 10 validation folds for different prediction horizons. Multiple model architectures and hyperparameter configurations were evaluated. The table shows the mean skill score across all folds, where the skill score measures performance relative to a naive predictor. A score of* ≥ 0 *indicates performance equal to or better than the naive predictor, while values closer to 1 indicate increasingly better relative performance.*

THCS: Models and Average Test Skill Scores

Horizon (minutes)	**Transformer** 32-4-64-2-0.1 19k params ITS	**Transformer** 128-4-256-2-0.1 271k params ITS	**LSTM** 128-2-0.1-FFN64 223k params ITS
0.5	0.5340	0.5085	0.5680
1	0.6997	0.6711	**0.7227**
5	0.7981	0.8012	**0.8091**
10	0.7875	0.7943	**0.8095**
15	0.7703	0.7798	**0.7879**

Horizon (minutes)	**Transformer** 32-4-128-2-0.1 19k params Single step training	**Transformer** 128-4-256-2-0.1 271k params Single step training	**LSTM** 128-2-0.1-FFN64 223k params Single step training
0.5	0.5522	0.5605	**0.5742**
1	0.6993	0.7034	0.7145
5	0.7761	0.7795	0.7880
10	0.7601	0.7685	0.7710
15	0.7386	0.7510	0.7452

Table 5.2 *Average THCS test skill scores across k = 10 validation folds for different prediction horizons and model configurations, evaluated relative to a naive predictor.*

Table 5.3 *Prediction skill statistics on test data for artificial systems under varying disturbance levels. The same ten artificial systems were used in all experiments, while the noise level applied during data simulation was varied. A Transformer-128-4-256-2-0.1 model (271k parameters) trained with the ITS was used in all cases.*

Transformer-128-4-256-2-0.1

Noise:	$\sigma_{\text{load}} = 0$ & $\sigma_{\text{noise}} = 0$					$\sigma_{\text{load}} = 0.1$ & $\sigma_{\text{noise}} = 0.01$				
Horizon (steps)	Mean	Std	Min	Max	N	Mean	Std	Min	Max	N
1	0.902	0.072	0.719	0.968	10	0.686	0.120	0.387	0.830	10
2	0.931	0.062	0.763	0.979	10	0.767	0.119	0.447	0.879	10
10	0.977	0.041	0.861	0.996	10	0.885	0.117	0.555	0.951	10
20	0.981	0.042	0.861	0.999	10	0.903	0.142	0.501	0.968	10
30	0.982	0.047	0.849	0.999	10	0.905	0.171	0.421	0.975	10

Noise:	$\sigma_{\text{load}} = 0.5$ & $\sigma_{\text{noise}} = 0.05$					$\sigma_{\text{load}} = 1$ & $\sigma_{\text{noise}} = 0.1$				
Horizon (steps)	Mean	Std	Min	Max	N	Mean	Std	Min	Max	N
1	0.331	0.041	0.250	0.404	10	0.075	0.181	-0.230	0.258	10
2	0.426	0.071	0.331	0.556	10	0.216	0.078	0.101	0.303	10
10	0.592	0.156	0.398	0.895	10	0.340	0.104	0.235	0.504	10
20	0.673	0.137	0.494	0.935	10	0.400	0.117	0.285	0.563	10
30	0.723	0.118	0.570	0.948	10	0.450	0.115	0.336	0.609	10

Table 5.4 *Prediction skill statistics on test data for artificial systems under varying disturbance levels. The same ten artificial systems were used in all experiments, while the noise level applied during data simulation was varied. An LSTM-128-2-0.1-FFN64 model (223k parameters) trained with the ITS was used in all cases.*

LSTM-128-2-0.1-FFN64

Noise:	$\sigma_{\text{load}} = 0$ & $\sigma_{\text{noise}} = 0$					$\sigma_{\text{load}} = 0.1$ & $\sigma_{\text{noise}} = 0.01$				
Horizon (steps)	Mean	Std	Min	Max	N	Mean	Std	Min	Max	N
1	0.954	0.028	0.904	0.987	10	0.751	0.073	0.617	0.864	10
2	0.968	0.021	0.929	0.991	10	0.824	0.055	0.710	0.906	10
10	0.993	0.006	0.979	0.998	10	0.929	0.020	0.893	0.955	10
20	0.996	0.004	0.984	0.999	10	0.954	0.015	0.922	0.969	10
30	0.997	0.004	0.986	1.000	10	0.963	0.014	0.932	0.976	10

Noise:	$\sigma_{\text{load}} = 0.5$ & $\sigma_{\text{noise}} = 0.05$					$\sigma_{\text{load}} = 1$ & $\sigma_{\text{noise}} = 0.1$				
Horizon (steps)	Mean	Std	Min	Max	N	Mean	Std	Min	Max	N
1	0.298	0.049	0.232	0.380	10	0.234	0.049	0.150	0.310	10
2	0.373	0.057	0.309	0.487	10	0.304	0.064	0.188	0.408	10
10	0.581	0.081	0.454	0.717	10	0.616	0.116	0.374	0.754	10
20	0.683	0.075	0.553	0.803	10	0.710	0.107	0.483	0.831	10
30	0.731	0.066	0.611	0.838	10	0.745	0.094	0.551	0.860	10

6

Conclusions

The purpose of the study was to investigate whether models of large-scale control systems at ESS could be created. Specifically, it aimed to obtain data-driven models for the CMS and the THCS. To achieve this, the thesis had to answer the research questions of how a modeling pipeline could be structured, what ML methods were applicable and what limitations existed for the systems.

To answer the first question, a modeling pipeline was constructed primarily based on Ljung [4]. To accommodate the control aspect of the systems, a way of structuring the models in a closed-loop setting was formulated. The models were structured as autoregressive one-step predictors capable of handling both endogenous and exogenous variables, which in deployment would enable them to be run as simulators given new control laws and closed-loop feedback.

Furthermore, the Iterative Training Strategy (ITS) was introduced to help the models remain accurate during multi-step predictions. This strategy proved useful for long-horizon performance throughout all models and systems. It can therefore be noted that this contribution could prove useful in future work involving closed-loop autoregressive models for multi-step prediction performance.

To answer the second question, both the Transformer and LSTM models were implemented and evaluated. These modern architectures have the ability to capture large-scale, complex and temporal dynamics. Given a similar model size, both architectures demonstrated similar performance and were therefore concluded to be suitable for the modeling task.

The results indicate that the implemented modeling approach looks promising, achieving solid performance on both the THCS and the artificial systems.

Performance on the CMS was slightly worse, although the models still outperformed the naive predictor on average. To answer the third research question, a likely limiting factor for the CMS performance was the limited amount of informative and sufficiently varied operational data available. Still, as the models performed well on both the THCS and the artificial systems, the results indicate that the models seem capable of capturing complex dynamics given a sufficient amount of data and that they provide feasible alternatives for the task. To more fully assess the achievable accuracy of the models, especially for the CMS, additional data will be required.

Apart from limited qualitative data being a limiting factor, the study also did not rigorously investigate the performance of the models operating in completely new regimes. Although the ITS enabled multi-step evaluation on left-out clusters, there is no guarantee that the performance generalizes to new and atypical control laws or operating scenarios. Consequently, the feasibility of using the models in a simulative setting reliably for out-of-domain situations requires further investigation.

In conclusion, the findings suggest several implications for further development. The results indicate that data-driven approaches represent a feasible alternative for modeling large-scale control systems. Specifically, the closed-loop simulation capabilities could offer a safe environment for operator training, system analysis, and initial control law evaluation. These outcomes provide a baseline suggesting that machine learning models may be increasingly relevant to control-system environments.

6.1 Future Work:

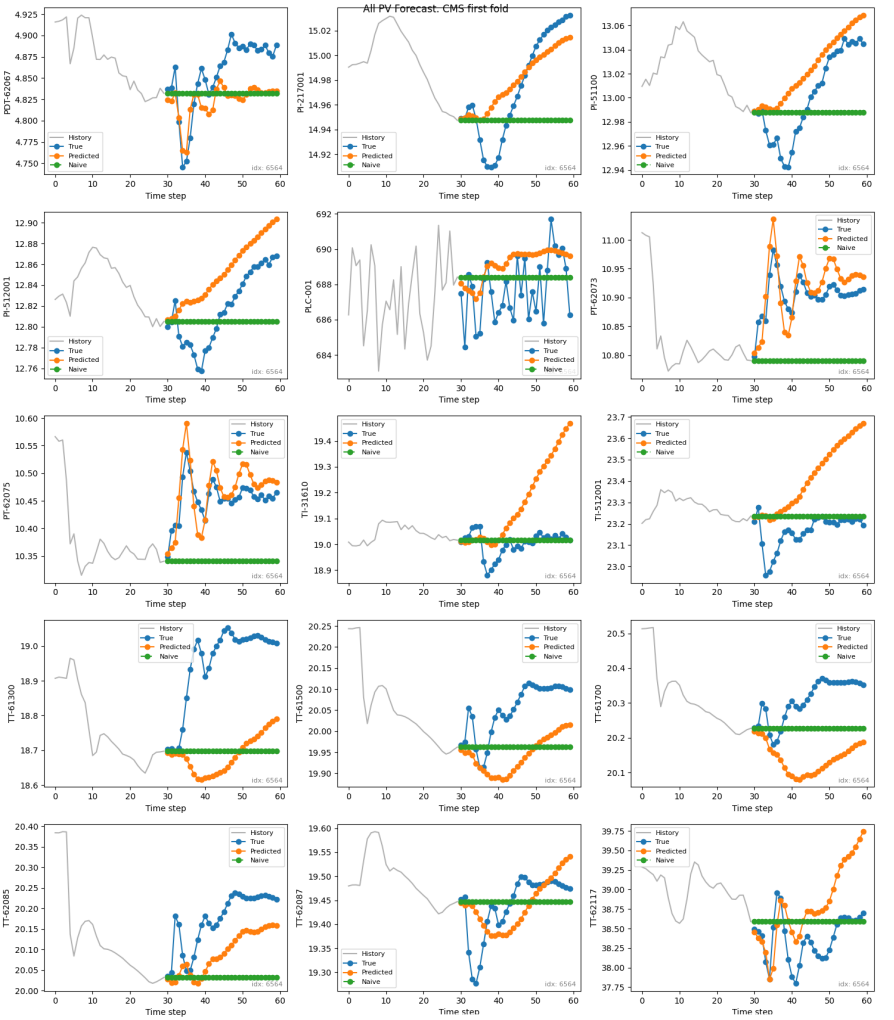
Additional data from the CMS and the THCS will become available during continued system operation, especially for the CMS, as planned beam-on-target operation will generate substantially more informative, system-exciting data. It will then be possible to more rigorously evaluate both the upper-bound performance and the generalization capabilities of the proposed modeling framework. In particular, future work could investigate:

- A natural and needed extension of this work is to evaluate how well the models perform in out-of-domain operating scenarios and under previously unseen control laws. Investigating these scenarios would provide deeper insight into the fundamental limitations of the modeling strategy independent of data availability. Furthermore, it would help determine the extent to which accurate, full-scale digital-twin simulators can be reliably deployed for control systems at ESS.

- Integration of models at ESS. Further investigation could assess the feasibility of utilizing the model for actual control of live systems by leveraging predictions of system trajectories.
- The potential for using predicted trajectories to identify and alarm for deviations from nominal system operation could also be explored.
- Another possible direction for future work would be to look at the proposed ITS from a mathematical and theoretical perspective, to try to understand the mechanism behind it and whether it actually acts as an effective regularizer or not.

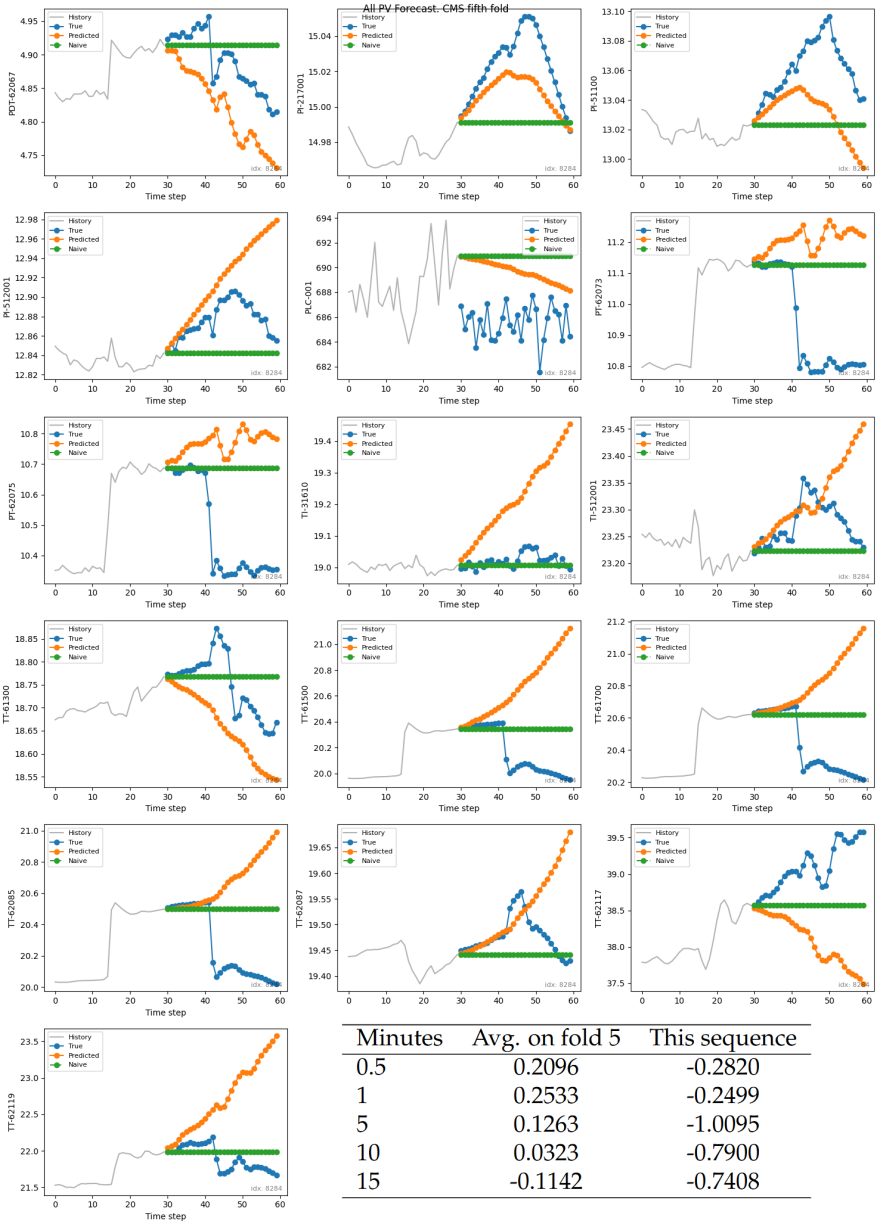
7

Appendix – Simulated Sequences



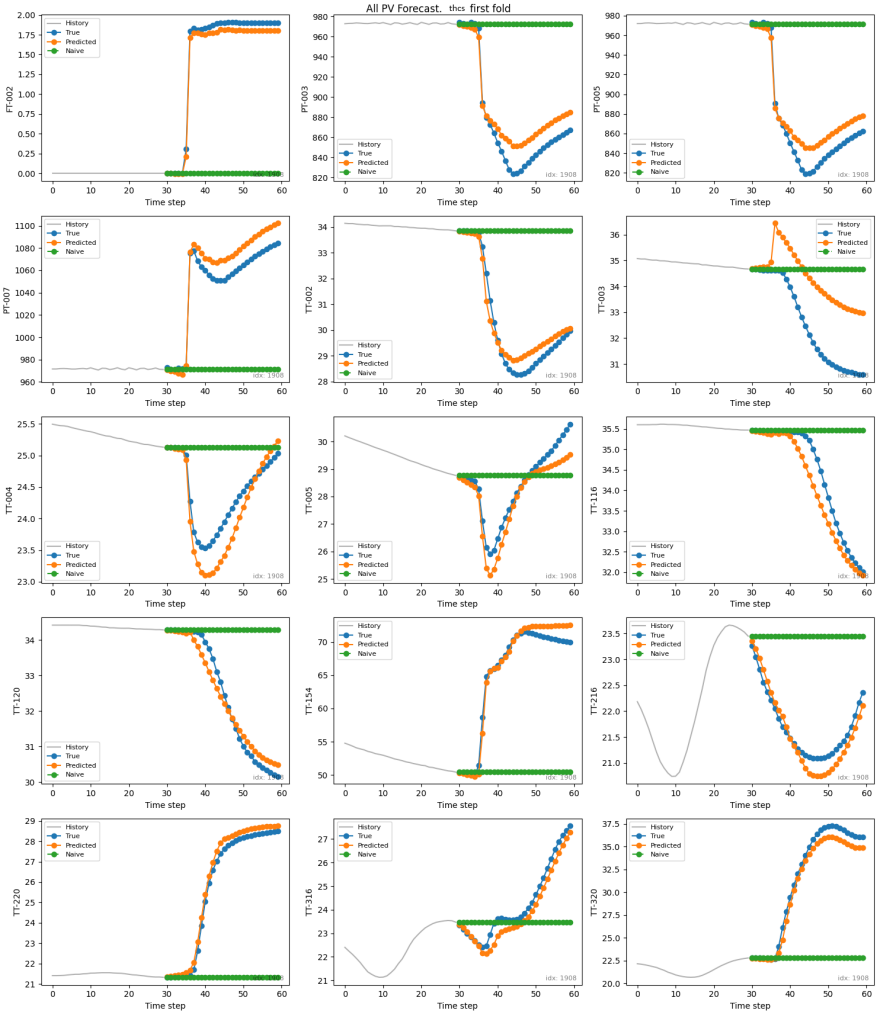
Minutes	Avg on fold 1	This sequence
0.5	0.3728	-0.1613
1	0.4922	0.5168
5	0.5643	-0.0404
10	0.5429	0.6959
15	0.5219	0.7923

Figure 7.1 CMS: a well-performing simulated 15 minute sequence, with skill scores below.. The plot shows all simulated target PV trajectories for a specific sequence during a selected high-variance period, avoiding intervals where the system inputs were stale. Transformer 128-4-256-2-0.1 model evaluated on test fold 1, which was a fold were the model had good performance.



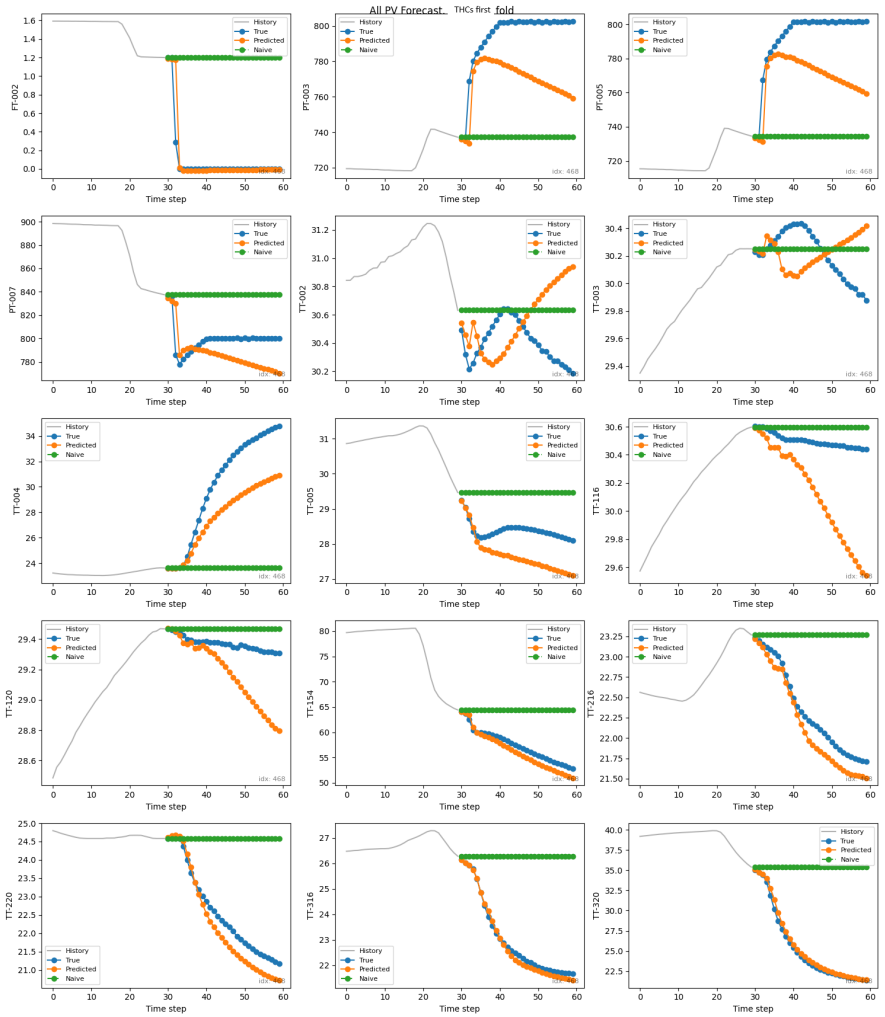
Minutes	Avg. on fold 5	This sequence
0.5	0.2096	-0.2820
1	0.2533	-0.2499
5	0.1263	-1.0095
10	0.0323	-0.7900
15	-0.1142	-0.7408

Figure 7.2 CMS: a worse performing simulated 15 minute sequence, with skill scores below. The plot shows all simulated target PV trajectories for a specific sequence during a selected high-variance period, avoiding intervals where the system inputs were stale. Transformer 128-4-256-2-0.1 model evaluated on test fold 5. Fold 5 gave really poor results, hence the bad performance of the model trying to simulate a sequence given that test data.



Minutes	Avg on fold 1	This sequence
0.5	0.7776	0.5758
1	0.9026	0.7935
5	0.9523	0.9962
10	0.9445	0.9939
15	0.9331	0.9923

Figure 7.3 THCS: a really good performing simulated 15 minute sequence, with skill scores below.. The plot shows all simulated target PV trajectories for a specific sequence during a selected high-variance period, avoiding intervals where the system inputs were stale. LSTM 128-2-0.1-FFN64 model evaluated on test fold 1, which was a fold were the model had good performance.



Minutes	Avg on fold 1	This sequence
0.5	0.7776	0.8246
1	0.9026	0.8354
5	0.9523	0.9849
10	0.9445	0.9613
15	0.9331	0.9540

Figure 7.4 THCS: an well-performing simulated 15 minute sequence, with skill scores below. The plot shows all simulated target PV trajectories for a specific sequence during a selected high-variance period, avoiding intervals where the system inputs were stale. LSTM 128-2-0.1-FFN64 model evaluated on test fold 1, which was a fold were the model had good performance.

Bibliography

[1] European Spallation Source ERIC. *Science using neutrons*. 2026. URL: https://ess.eu/science-using-neutrons (visited on 2026-04-08).

[2] European Spallation Source ERIC. *Instruments*. 2026. URL: https://ess.eu/instruments (visited on 2026-04-08).

[3] European Spallation Source ERIC. *Ess achieves beam dump, accelerator commissioning underway*. 2025. URL: https://ess.eu/article/2025/05/16/ess-achieves-beam-dump-accelerator-commissioning-underway (visited on 2026-04-08).

[4] L. Ljung. *System Identification: Theory for the User*. 2nd ed. Prentice Hall PTR, Upper Saddle River, NJ, 1999. ISBN: 978-0136566953.

[5] J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, P.-Y. Glorennec, H. Hjalmarsson, and A. Juditsky. "Nonlinear black-box modeling in system identification: a unified overview". *Automatica* **31**:12 (1995), pp. 1691–1724. DOI: 10.1016/0005-1098(95)00120-8.

[6] L. Ljung, C. Andersson, K. Tiels, and T. B. Schön. "Deep learning and system identification". *IFAC-PapersOnLine* **53**:2 (2020). 21st IFAC World Congress, pp. 1175–1181. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2020.12.1329.

[7] S. Hochreiter and J. Schmidhuber. "Long short-term memory". *Neural Computation* **9**:8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention is all you need". *arXiv preprint* (2017). arXiv: 1706.03762 [cs.CL].

[9] Wikipedia contributors. *Recurrent neural network*. 2026. URL: https://en.wikipedia.org/w/index.php?title=Recurrent_neural_network (visited on 2026-03-30).

[10] W. Commons. *File:nn lstm-cell v2.svg — wikimedia commons, the free media repository*. [Online; accessed 17-May-2026]. 2026. URL: https://commons.wikimedia.org/wiki/File:NN_LSTM-Cell_v2.svg.

[11] PyTorch Developers. *torch.nn.LSTM*. PyTorch. 2026. URL: https://docs.pytorch.org/docs/stable/generated/torch.nn.LSTM.html (visited on 2026-03-04).

[12] Wikipedia contributors. *Gated recurrent unit*. 2025. URL: https://en.wikipedia.org/w/index.php?title=Gated_recurrent_unit (visited on 2026-03-30).

[13] PyTorch Contributors. *LayerNorm — PyTorch 2.11 Documentation*. Accessed: 2026-05-05. 2026. URL: https://docs.pytorch.org/docs/2.11/generated/torch.nn.LayerNorm.html.

[14] C. Gyllenstierna and V. Ritseson. *Streamlining Data-Centric ML-Ops in the Integrated Control System at ESS*. MSc thesis TFRT-6278. Lund University, Department of Automatic Control, Lund, Sweden, 2025.

[15] B. Lim, S. O. Arik, N. Loeff, and T. Pfister. "Temporal fusion transformers for interpretable multi-horizon time series forecasting". *arXiv preprint* (2020). arXiv: 1912.09363. URL: https://arxiv.org/abs/1912.09363.

[16] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski. "Deepar: probabilistic forecasting with autoregressive recurrent networks". *International Journal of Forecasting* **36**:3 (2020), pp. 1181–1191. DOI: 10.1016/j.ijforecast.2019.07.001.

[17] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan. "Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting". In: *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*. Vancouver, Canada, 2019. arXiv: 1907.00235 [cs.CL].

[18] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Ré. "Hippo: recurrent memory with optimal polynomial projections". *arXiv preprint* (2020). arXiv: 2008.07669 [cs.LG].

[19] A. Gu, K. Goel, and C. Ré. "Efficiently modeling long sequences with structured state spaces". *arXiv preprint* (2022). arXiv: 2111.00396 [cs.LG].

[20] M. Forgione, F. Pura, and D. Piga. "From system models to class models: an in-context learning paradigm". *IEEE Control Systems Letters* **7** (2023), pp. 3513–3518. DOI: 10.1109/LCSYS.2023.3335036.

[21] C. Källström. *Data Quality and Quantity for Machine Learning at the European Spallation Source*. Master's Thesis LU-CS-EX: 2025-54. Department of Computer Science, Lund University, 2025.

[22] EPICS Collaboration. *About epics*. Accessed: 9 April 2026. 2024. URL: https://epics-controls.org/about-epics/.

[23] PyTorch Contributors. *A gentle introduction to torch.autograd — PyTorch tutorials 2.12.0+cu130 documentation*. https://docs.pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html. Accessed: 2026-05-19. 2026.

[24] Wikipedia contributors. *Forecast skill — Wikipedia, the free encyclopedia*. [Online; accessed 12-May-2026]. 2026. URL: https://en.wikipedia.org/w/index.php?title=Forecast_skill&oldid=1346513733.

[25] Wikipedia contributors. *Cross-validation (statistics) — Wikipedia, the free encyclopedia*. [Online; accessed 5-May-2026]. 2026. URL: https://en.wikipedia.org/w/index.php?title=Cross-validation_(statistics)&oldid=1349470997.

[26] PyTorch Contributors. *Torch.optim.adamw*. 2024. URL: https://docs.pytorch.org/docs/stable/generated/torch.optim.AdamW.html (visited on 2026-04-09).

[27] Wikipedia contributors. *Jordan normal form — Wikipedia, the free encyclopedia*. [Online; accessed 11-May-2026]. 2026. URL: https://en.wikipedia.org/w/index.php?title=Jordan_normal_form&oldid=1351878036.

[28] Wikipedia contributors. *Qr decomposition — Wikipedia, the free encyclopedia*. [Online; accessed 4-May-2026]. 2026. URL: https://en.wikipedia.org/w/index.php?title=QR_decomposition&oldid=1347123102.

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
June 2026

Document Number
TFRT-6313

Author(s)
Eskil Olofsson

Supervisor
Karin Rathsman, ESS, Sweden
Attila Horvath, ESS, Sweden
Johan Eker, Dept. of Automatic Control, Lund University, Sweden
Tore Hägglund, Dept. of Automatic Control, Lund University, Sweden
Bo Bernhardsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle

Data-Driven Modeling of Large-Scale Control Systems at the European Spallation Source (ESS)

Abstract

Control systems at European Spallation Source (ESS) are large-scale, complex and difficult to model using traditional approaches. Data-driven methods are therefore required to model systems such as the Cryogenic Moderator System (CMS) and Target Helium Cooling System (THCS). This work presents a control-system modeling approach based on the Transformer and Long Short-Term Memory (LSTM), enabling long-range multivariate predictions using control variables as exogenous inputs. The proposed framework integrates these models into a closed-loop setting, enabling iterative, autoregressive, predictions during both training and deployment. The results were promising for the THCS and artificial systems, while performance on the CMS was more limited. This discrepancy was likely caused by insufficient system excitation in the available datasets, owing to the absence of beam-on-target operations at ESS to date. While the proposed data-driven framework demonstrates considerable potential, its overall reliability is currently constrained by the limited diversity of operational data and a lack of out-of-domain testing.

Keywords
data-driven modeling, control systems, Transformers, LSTM, NARX

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-83

Recipient's notes

Security classification

http://www.control.lth.se/publications/