SoftwareX 30 (2025) 102148

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

MELGYM: A dynamic control interface for MELCOR simulations

Antonio Manjavacas [a,*], Juan Gómez-Romero [a], Damien Ernst [c], Manuel A. Vázquez-Barroso [b], Francisco Martín-Fuertes [d]

[a] Department of Computer Science and Artificial Intelligence, Universidad de Granada, Granada, Spain
[b] Department of Structural Mechanics and Hydraulic Engineering, Universidad de Granada, Granada, Spain
[c] Montefiore Institute, Université de Liège, Liège, Belgium
[d] Centre for Energy, Environmental and Technological Research (CIEMAT), Madrid, Spain

ARTICLE INFO

Keywords:
MELCOR
Gymnasium
Control
Reinforcement learning

ABSTRACT

MELCOR is a computer code for the simulation and safety analysis of nuclear facilities, comprising several modules for the calculation of thermal–hydraulic phenomena and aerosol physics. However, MELCOR offers limited support for interactive control, such as valve opening or flow rates redefinition, as its batch execution mode does not allow real-time modification of model parameters. To overcome this limitation, we present MELGYM, a Gymnasium-based interface that facilitates interactive control in MELCOR. MELGYM enables the training of reinforcement learning agents on MELCOR simulations, introducing real-time interaction and enabling the definition of reward functions based on simulation outcomes.

Code metadata

Current code version	v1.6.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-24-00640
Permanent link to Reproducible Capsule	–
Legal Code License	GNU General Public License
Code versioning system used	git
Software code languages, tools, and services used	python, melcor
Compilation requirements, operating environments & dependencies	gymnasium, numpy, matplotlib, melkit, pyyaml, tensorboard
If available Link to developer documentation/manual	https://melgym.readthedocs.io/
Support email for questions	manjavacas@ugr.es

1. Motivation and significance

Simulation tools play a pivotal role in the field of nuclear safety engineering. Nuclear codes facilitate the design and validation of critical infrastructures such as reactors [1], spent nuclear fuel plants [2], tokamaks [3], or particle accelerators [4]. Their operation generally involves the discretized computation of equations related to neutronics, thermal-hydraulics or aerosol physics, approximated through different numerical methods [5].

A variety of nuclear codes are available for severe accident analysis, each varying based on the computational methods used, the physical phenomena simulated, and the level of detail [6]. Examples include RELAP5, TRACE, MAAP, ASTEC, PARCS, and MELCOR.

MELCOR is an engineering-level computer code developed by Sandia National Laboratories for the U.S. Nuclear Regulatory Commission [7]. Its primary use is to model and study the progression of severe accidents in nuclear power plants, although it has evolved to address other types of accidents and facilities. Thus, MELCOR comprises a series of interrelated packages for different simulation-time computations, allowing the modelling of multiple physical phenomena. It has been employed in relevant projects [8,9] and counts with an extensive user base [10].

MELCOR simulations involve several interrelated elements, such as *control volumes* (CVs), *flow paths* (FLs) – which represent CV connections –, or *heat structures* (HSs), all defined by a set of physical properties

* Corresponding author.
E-mail addresses: manjavacas@ugr.es (Antonio Manjavacas), jgomez@decsai.ugr.es (Juan Gómez-Romero), dernst@uliege.be (Damien Ernst), manvazbar@ugr.es (Manuel A. Vázquez-Barroso), francisco.martin-fuertes@ciemat.es (Francisco Martín-Fuertes).

https://doi.org/10.1016/j.softx.2025.102148
Received 2 December 2024; Received in revised form 12 March 2025; Accepted 24 March 2025

and initial simulation conditions. MELCOR records the evolution of different variables, such as temperature, pressure, or gas concentrations, resulting from their simulation.

Additionally, users can employ *control functions* (CFs) to define functions of variables contained in the MELCOR database. CFs real or logical values are themselves variables that can be used by other model elements during simulation. For example, this allows the opening of an FL to be modified according to the pressure of a given CV, the laminar friction coefficient in an FL segment to be changed over time, or the rate of mass addition in a CV to be modified in response to a specific atmosphere change.

The control capabilities of MELCOR are largely defined by CFs. However, we identify a number of limitations in the control methods offered by MELCOR. Firstly, MELCOR does not support real-time interaction with simulations, limiting control to predefined functions that cannot be easily modified during execution due to its batch-mode operation. The default control alternatives are limited to user-defined functions, built-in Proportional–Integral–Derivative (PID) controllers, and hysteresis functions, which must be manually adjusted in a user-unfriendly way each time the simulation model is modified. Moreover, complex control logic requires the concatenation of several interdependent CFs, which can be an intricate process from the programmer's point of view, so more flexibility and code customization is desirable.

To address these shortcomings, we present MELGYM, a Python library designed to facilitate interactive control with MELCOR. MELGYM allows the definition of custom reactive controllers and facilitates interaction through alternative control paradigms, including reinforcement learning (RL). The tool is based on the Gymnasium interface [11], a standard in the RL domain that ensures compatibility with widely used RL libraries.

In recent years, machine learning solutions based on (Deep) Reinforcement Learning (DRL) have emerged as an outstanding competitor to conventional control methods, with numerous applications in Heating, Ventilation and Air Conditioning (HVAC) [12], flow control [13], or energy systems [14,15]. DRL comprises a series of data-driven algorithms that can handle control tasks in complex high-dimensional environments, offering generalization and low reliance on expert knowledge.

The aim of this work is to describe the main features and capabilities of the MELGYM ecosystem, showing its integration with DRL controllers. We also present a real use case in the safety design of the IFMIF-DONES particle accelerator facility [16]. To the best of our knowledge, no similar tool is publicly available, so its potential contribution to the MELCOR community is remarkable.

The remainder of the paper is structured as follows: Section 2 outlines the software architecture and main functionalities of MELGYM. Section 3 presents a case study of MELGYM in a real-world setting, while Section 4 outlines the key contributions of the tool. Finally, the conclusions of the paper are addressed in Section 5.

2. Software description

Prior to describing MELGYM, the relationship between the RL terminology and the problem addressed is provided. The typical formulation of RL problems is that of Markov Decision Processes (MDPs), where an *agent* interacts with a dynamic process, known as the *environment*, over a discrete sequence of time steps $\mathcal{T} = \{0, 1, 2, \dots\}$. As illustrated in Fig. 1, every MELGYM environment correspond to a simulated MELCOR model with which an agent interacts.

At each time step t, the agent perceives an *observation* $o_t \in \mathcal{O}$, including a subset of the variables that define the current *state* of the simulation $s_t \in \mathcal{S}$. Based on this information, an *action* $a_t \in \mathcal{A}(s)$ modifying the MELCOR model is performed. The updated model is then simulated during a specified period of time, resulting in a new state $s_{t+1} \in \mathcal{S}$ from which a *reward* signal $r_t \in \mathbb{R}$ is computed. This reward

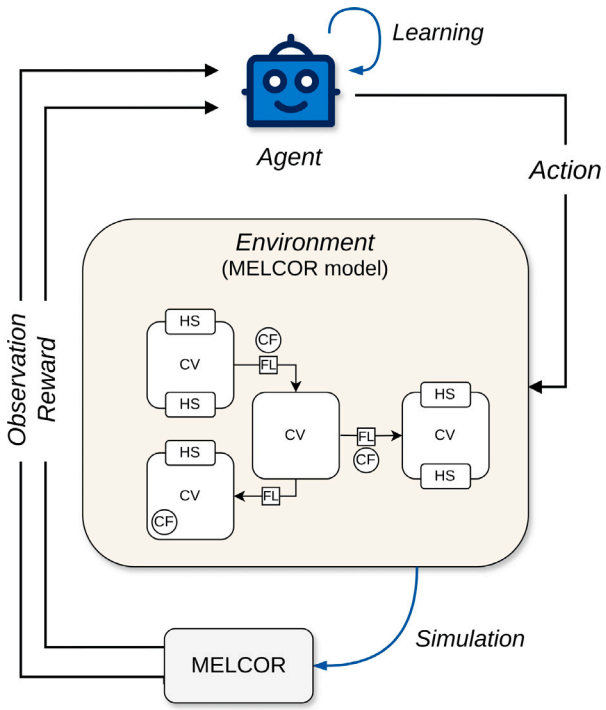

Fig. 1. Agent–environment interaction in MELGYM.

represents the goodness of the resulting transition, thus guiding the agent's learning process.

The mapping between states and actions is defined by a policy function π, such that $a_t \sim \pi(\cdot|s_t)$. The goal of an RL agent is to learn an optimal policy π^* allowing the maximization of cumulative rewards over time, that is, the maximization of $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$, for any t, where $\gamma \in [0, 1]$ is a discount factor used to weight future rewards.

Combining RL with neural networks gives rise to *deep* reinforcement learning algorithms, which assume a parameterized policy π_θ – e.g., a feed-forward neural network with weights θ – whose expected return is progressively approximated to that of an optimal policy by applying gradient-based optimization.

The main contribution of MELGYM is to enable the integration of DRL algorithms in MELCOR simulations. As opposed to traditional controllers, these algorithms do not require the common manual tuning of controllers such as PIDs. However, MELGYM is not exclusively restricted to DRL control, allowing the definition of any control logic that fits the agent–environment interaction loop.

In control problems, we consider a set of *control signals* and *output variables* corresponding to the controlled system. In the case of MELGYM, control signals are user-defined variables, such as the flow rate through a given path or the opening/closing of a valve, that are set by the controller to influence the state of the system. On the other hand, output variables reflect the actual state of the controlled system — e.g., volumes' pressures, temperatures or gas concentrations. Thus, guiding an autonomous controller to properly achieve the desired outputs – e.g., around a given setpoint – requires the user to specify a reward function that correctly reflects the effectiveness of the control actions performed.

2.1. Software architecture

When implementing RL environments, it is a standard practice to adhere to the Gymnasium interface [11]. This programming framework specifies a set of classes and functions required to simulate the typical RL agent–environment interaction. It offers a range of

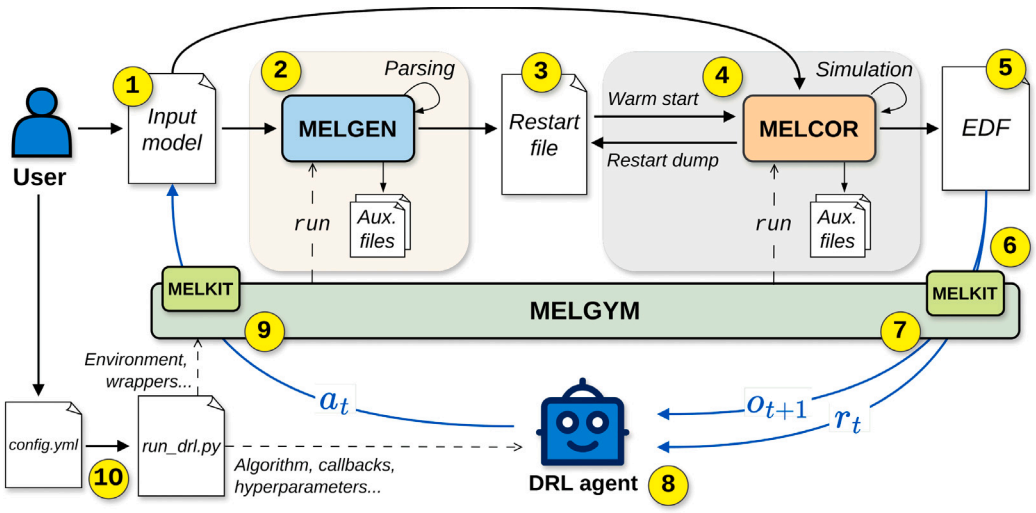

Fig. 2. MELGYM overview.

tools for defining custom rewards and wrappers, as well as classic control environments. Moreover, Gymnasium allows the creation of custom environments with user-defined state and action spaces. As a Gymnasium-based tool, MELGYM adheres to its standard structure, ensuring usability and compatibility with widely-used RL libraries, such as Stable-Baselines3 [17].

Fig. 2 shows an overview of the MELGYM ecosystem. An explanation of the different elements represented is provided below:

(1). The *input model* is the file containing the MELCOR model to be simulated. Models are divided in two different sections: a) the MELGEN input, where the elements that make up the model are defined – e.g., CVs, FLs, HSs, CFs –, and b) the MELCOR input, where the simulation configuration is specified — e.g., simulation time, file naming, log frequency, termination conditions.

(2). MELGEN is the parser used to verify the syntax and grammar correctness of the model file. It is a complementary executable to MELCOR that outputs a series of diagnostic files and prepares the model for execution. Therefore, it is only executed once.

(3). If the format of the input model file is correct, the MELGEN execution results in a *restart file* where the initial conditions of the simulation are recorded. This is a binary file where MELCOR periodically dumps information, allowing the simulation to be resumed from any time step.

(4). Once the restart file is generated, the MELCOR simulation begins. Firstly, the simulator performs a *warm start*, reading the current state of the simulation from the restart file. Subsequently, a series of simulation steps are performed, and the resulting state of the simulation is saved in the restart file. Additional logs with the evolution of different variables of the MELCOR database are equally written in auxiliary files.

(5). The *external data file* (EDF) is the file used by default to register the evolution of the controlled variables. The recorded values are used to compute a reward signal for the DRL agent.

(6). Reading the EDF and updating the input model is not done directly by MELGYM, but by means of the MELKIT utility. This is an auxiliary tool that enables the manipulation of MELCOR files, enabling typical CRUD operations via Python classes and methods.

(7). Once the current state of the controlled variables has been read from the EDF, MELGYM processes this information to generate the associated reward signal r_t and provide the agent with an observation of the environment o_{t+1} that allows it to act accordingly.

(8). The DRL agent can be any type of agent specified by the user or provided by a Gym-compatible DRL library, such as Stable-Baselines3. The agent receives information about the current state of the controlled variables and generates an action that modifies the input model.

(9). The agent's actions are passed back through the MELGYM/MELKIT interface, which translates them into the corresponding change on the input model file. Generally, actions are normalized, so it is necessary to denormalize them, search in the input model for the element to be modified, and apply the corresponding change.

(10). This process is repeated during a number of episodes defined by the user in the `config.yml` configuration file. This file is taken as an input by the `run_drl.py` executable, which facilitates the training and testing of DRL agents, automating the definition of wrappers, callbacks, algorithm hyperparameters, and environment settings.

It can be noted that the operation of MELGYM is primarily based on the discretization of simulations from intermediate warm starts, between which control actions are executed. Appendix includes a detailed example demonstrating the equivalence between warm start-based control and the default control offered by MELCOR.

2.2. Software functionalities

To show the functionalities of MELGYM, we rely on the code in Listing 1, which represents a basic configuration setup for automatically train a DRL agent with MELGYM.

Listing 1 Sample `yaml` configuration file

```yaml
id: example
tasks:
    - train
    - test
env:
    name: branch_1
    params:
        max_deviation: 40
        check_done_time: 500
        control_horizon: 10
        render_mode: 'pressures'
algorithm:
    name: TD3
    params:
        batch_size: 128
        learning_rate: 0.001
    train_params:
```

```
        total_timesteps: 50000
        eval_freq: 10000
        n_eval_episodes: 1
wrappers:
    - norm_obs
callbacks:
    - EvalCallback
    - TbMetricsCallback
    - EpisodicDataCallback
paths:
    tensorboard_dir: './tensorboard/'
    best_models_dir: './best_models/'
    ep_metrics_dir: './ep_metrics/'
```

Task definition. Each DRL execution requires an identifier (*line 1*) and the specification of the tasks to be performed (*lines 2–4*). In this case, the DRL agent is first trained, and then the best model version obtained is tested.

Environment configuration. We then define the MELGYM environment in which the agent will be executed (*lines 5–6)*. In this case it is `branch_1`, an example already implemented in MELGYM, where we seek to achieve optimal pressure control in different volumes by adjusting the flow rate of an FL. A series of environment parameters are also indicated, including the maximum deviation allowed with respect to the initial pressure of the volumes (*line 8*), the timestep from which an episode can be truncated (*line 9*), the number of simulation steps between actions (*line 10*), and the rendering mode (*line 11*).

Agent configuration. The following fields contain the configuration corresponding to the agent (*line 12*). It is specified that the DRL algorithm used is TD3 [18], as well as the hyperparameters of the algorithm (*lines 13–16*). On the other hand, the training parameters include the number of timesteps to be used for training (*line 18*), how often the agent will be evaluated (*line 19*), and the number of evaluation episodes (*line 20*).

Wrappers and callbacks. Within the wrappers configuration, a wrapper is used to normalize observations (*lines 21–22*). We also specify the use of three callbacks for: periodic agent evaluation (*line 24*), TensorBoard metrics logging (*line 25*), and custom episode-to-episode metrics logging (*line 26*). Finally, we indicate the paths where the TensorBoard data (*line 28*), the final obtained model (*line 29*), and the episodic logs (*line 30*) will be saved.

2.3. Sample code snippet analysis

Listing 2 shows a sample program where the training and evaluation of a TD3 agent on the MELGYM environment `branch_1` is performed. First, the environment is instantiated and its configuration specified. Subsequently, the agent is defined, trained and evaluated.

Listing 2 Sample TD3 control in a MELGYM environment

```
import melgym
import gymnasium as gym
from stable_baselines3 import TD3

env = gym.make('branch_1', render_mode=
'pressures')

# Training
agent = TD3('MlpPolicy', env)
agent.learn(total_timesteps=10_000)

# Evaluation
obs, info = env.reset()
done = trunc = False

while not (done or trunc):
    env.render()
    act, _ = agent.predict(obs)
    obs, rew, done, trunc, info = env.step
    (act)

env.close()
```

At the beginning of each episode, the environment is reset and the first observation is obtained. From this point on, as long as the termination or truncation conditions are not met, the agent gets and actions from its policy and performs a step in the environment. At each time step, the current state of the environment is rendered. When the termination or truncation condition is met, the execution ends and the environment is closed.

3. Illustrative example

The International Fusion Materials Irradiation Facility-DEMO Oriented Neutron Source (IFMIF-DONES) [16] will operate a continuous wave deuteron linear accelerator to irradiate sample materials under fusion-like conditions. Its objective is to provide relevant data for the construction of future nuclear fusion reactors.

As a radiological facility, the design of IFMIF-DONES must ensure the safety of the public, the environment and personnel through the implementation of appropriate safety measures [19], such as dynamic confinement. This measure involves maintaining higher-risk areas at a lower pressure than their surroundings to prevent the release of gases or other hazardous substances in the event of an accident.

In recent years, work on the safety design of the IFMIF-DONES main building has been associated with the development of multiple MELCOR models of its different subsystems [20]. In particular, a specific simulation model has been developed to analyse the ability of the nuclear HVAC system to ensure the dynamic confinement of the facility [19].

On this basis, we rely on MELGYM to assist in the design and analysis of the dynamic confinement measures, providing estimates of the airflow rates required by each of the HVAC branches. This enables designers to efficiently study different design alternatives, modifying possible room tightness or target pressures, and studying their impact on the consumption of the HVAC system.

The proposed control problem involves finding a control policy that guarantees the required air supply for an HVAC branch as a function of the different room pressures. Given imposed exhaust flow rates for each room – which are dependent on their corresponding confinement level –, and uncontrollable inter-room leaks, pressures must be kept stable within some given safety limits. As shown in Fig. 3, we represent the building rooms as CVs, and the connections between them – i.e., HVAC ducts – as FLs.

Formally, let P_{v_t} be the pressure of a given CV $v \in \mathbb{N}$ at time step t. Each observation $o_t \in \mathcal{O}$ includes the current pressures of each of the n CVs of the branch fed by the HVAC system, that is: $o_t = (P_{1_t}, P_{2_t}, \dots, P_{n_t})$. An action $a_t \in [0, 10]$ (m/s) is a continuous value representing the set value for the branch inlet air flow velocity. The objective is to determine the optimal air inlet so that all CVs remain at a stable pressure within specified safety limits. This is implicitly specified by a reward signal $r_t \in \mathbb{R}$ representing the sum of the distances from each room pressure to the mean value between its maximum and minimum safety limits (P_v^{max}, P_v^{min}), the sum of which we seek to minimize. It is defined as $r_t = \sum_{v=1}^{n} d\left(P_{v_t}, \frac{P_v^{max}+P_v^{min}}{2}\right)$.

We address this problem by using a similar code to the one presented in Listing 1 to train a TD3 agent in the `branch_2` environment, which models an HVAC branch serving 9 rooms. A simplified representation of this model is shown in Fig. 4.

The evaluation of the trained agent leads to the pressures evolution depicted in Fig. 5. It can be observed how the agent is able to keep the pressures stable within the corresponding safety limits. These results provide an estimate of the air inlet flow rate required by the

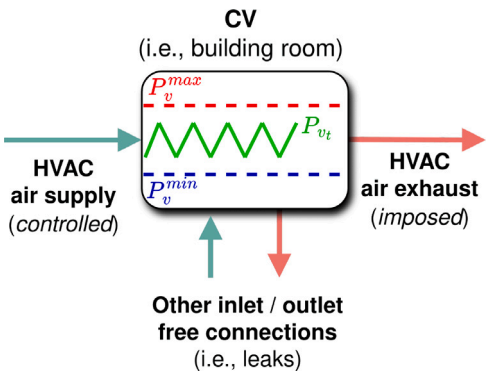

Fig. 3. MELCOR modelling of a building room. Each room is represented by a CV, whose pressure is monitored and kept within specified upper and lower limits. HVAC connections and leaks are represented by FLs with controlled, imposed or free flow rates.

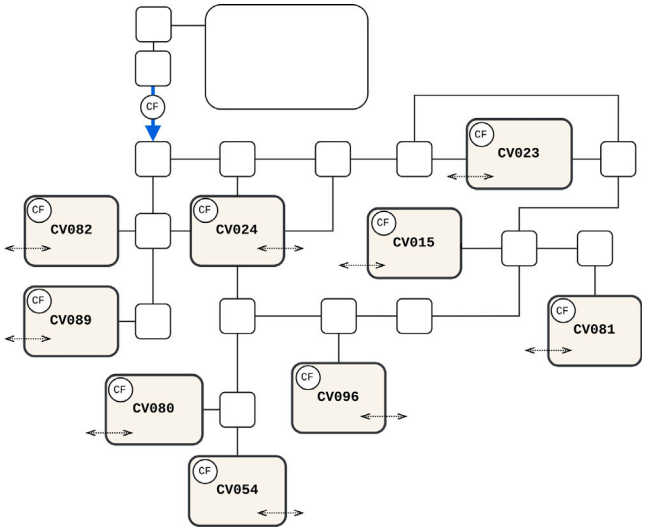

Fig. 4. Simplified diagram of the `branch_2` environment. Coloured CVs represent HVAC-served rooms. FLs are represented by lines. Auxiliary CVs representing duct junctions are shown in white. Each room CV includes a CF acting as a mass sink, emulating constant air extraction. Dotted arrows represent inlet and outlet leaks. The blue arrow corresponds to the air inlet source that is controlled by the agent. For more details, refer to the source code of this model at https://github.com/manjavacas/melgym/blob/main/melgym/data/branch_2.inp.

HVAC branch. This enables safety analysts to determine whether the confinement criteria are met, and to test alternative configurations of the model without the need for manual adjustment of the inlet flow rates.

4. Impact

MELGYM is a tool designed to enable user interaction and control over MELCOR simulations, allowing the definition of external controllers and interactive variable monitoring. MELGYM streamlines the definition of these controllers by leveraging widely adopted programming languages, such as Python. Furthermore, it enables the coupling of DRL agents on MELCOR simulations which, to the best of our knowledge, has not yet been explored in the literature.

The MELKIT submodule can also be used as an independent tool to manipulate MELCOR models in a simple and automated way. These tools seek to fulfil the objectives of previously abandoned projects, with significant applicability for the MELCOR community.

MELGYM is being a valuable addition to the safety design of IFMIF-DONES, receiving significant feedback from expert users who work with MELCOR on a daily basis. This provides the opportunity to develop new, customized environments where other variables can be controlled or monitored.

5. Conclusions

In this paper, we presented MELGYM, a Gymnasium-based tool that enables the definition of external controllers that interact with MELCOR simulations. The main feature of MELGYM is the ability to implement DRL-based controllers, allowing an adaptive control that takes advantage of machine learning capabilities.

The tool is currently being used in a relevant project such as IFMIF-DONES, making a significant contribution to the MELCOR ecosystem. As future work, it is planned to extend the number of environments offered by the tool, either for the needs of the IFMIF-DONES project or at the request of the MELCOR community.

CRediT authorship contribution statement

Antonio Manjavacas: Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Juan Gómez-Romero:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Conceptualization. **Damien Ernst:** Writing – review & editing, Validation, Supervision, Formal analysis, Conceptualization. **Manuel A. Vázquez-Barroso:** Data curation, Formal analysis, Investigation, Methodology, Software, Writing – review & editing. **Francisco Martín-Fuertes:** Writing – review & editing, Validation, Supervision, Resources, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

Acknowledgement to Adrien Bolland for his help in the formalization of the problem.

This work has been funded by ERDF/Junta de Andalucía through the project SE2021 UGR IFMIF-DONES, by MICIU/AEI/10.13039/501100011033 through the SINERGY project (PID2021.125537 NA.I00), and by ERDF/Junta de Andalucía through the D3S project (P21.00247).

Appendix. Warm start-based control

In order to demonstrate the equivalence between warm start-based control and MELCOR default control, a test based on the `presscontrol`[1] environment of MELGYM will be performed. This environment employs the MELCOR model represented in Fig. A.6.

The model includes a volume with time-independent properties (`CV001`) and a volume with variable properties (`CV002`), connected by both inlet and outlet FLs (`FL001`, `FL002`). While the air outlet remains constant (1 *m/s*), we introduce time-dependent variations in the air inlet to observe changes in the pressure level of `CV002`.

First, we apply these variations using an embedded function in the code, employing the MELCOR Tabular Function (TF) package. The function, represented in Listing 3, modifies the flow rate of `FL001` depending on the current timestep. It is a piecewise defined linear function whose turning points are those indicated in the second (x-axis, time) and third (y-axis, flow velocity) columns.

[1] https://github.com/manjavacas/melgym/blob/main/melgym/data/presscontrol.inp

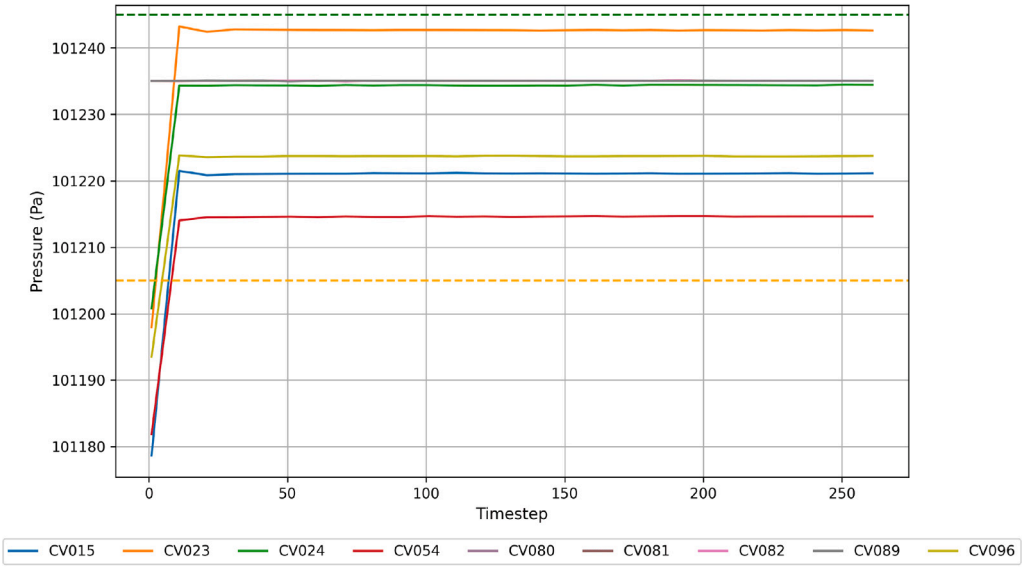

Fig. 5. Pressures controlled by a TD3 agent in the MELGYM `branch_2` environment. The agent manages to keep the pressures of all rooms – i.e., the pressures of the CVs – stable within the specified limits (dashed lines).

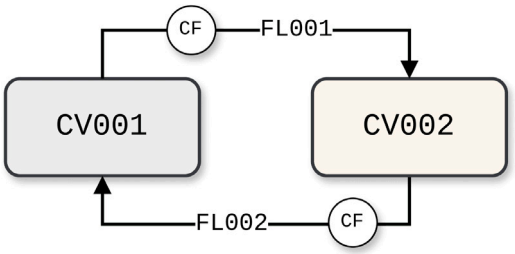

Fig. A.6. `presscontrol` MELCOR model.

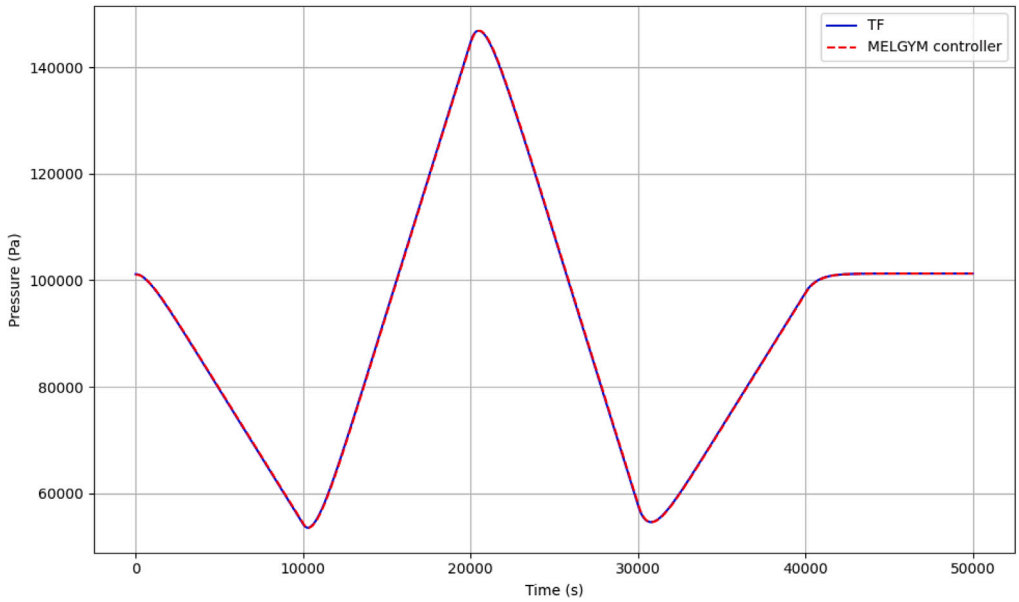

Fig. A.7. Comparison of pressure control in CV002 using a TF and an external controller based on MELGYM.

Listing 3 Time-dependent controller implemented in MELCOR

```
TF00100   VEL-IN    5     1.0
TF00110   0.0       1.0
TF00111   1e4       0.5
TF00112   2e4       1.5
TF00113   3e4       0.5
TF00114   4e4       1.0
```

Next, we define an external agent that implements the same function but leverages the MELGYM interface based on warm starts to perform this control, as shown in Listing 4.

Listing 4 Time-dependent controller based on the MELGYM interface

```
import melgym
import gymnasium as gym
import numpy as np

def get_action(time):
    # control turning points
    time_points = np.array([0, 1e4, 2e4, 3e4,
    4e4, 5e4])
    # normalized action values
    values = np.array([-0.8, -0.9, -0.7,
    -0.9, -0.8, -0.8])
    # interpolated action
    action = np.array(np.interp(time,
    time_points, values))
    return action

def test_controller(env):
    obs, _ = env.reset()
    done = truncated = False
    while not (done or truncated):
        action = get_action(info['time'])
        obs, reward, done, truncated, _ =
        env.step(action)
    env.close()

env = gym.make('presscontrol', max_tend=5e4,
               max_deviation=1e5,
               max_velocity=10,
               control_horizon=10)
test_controller(env)
```

The pressure evolution in CV002 for both cases is shown in Fig. A.7. It can be observed how warm starts do not imply differences compared to the control provided by a TF. In contrast, MELGYM offers the flexibility to employ a fully adaptable external control. Note that a sufficiently short control horizon (in this case, 10 timesteps) is necessary for accurate control, preventing significant delays between actions.

References

[1] Skolik K, Allison C, Hohorst J, Malicki M, Perez-Ferragut M, Pieńkowski L, Trivedi A. Analysis of loss of coolant accident without ECCS and DHRS in an integral pressurized water reactor using RELAP/SCDAPSIM. Prog Nucl Energy 2021;134:103648.

[2] Coindreau O, Jäckel B, Rocchi F, Alcaro F, Angelova D, Bandini G, Barnak M, Behler M, Da Cruz D, Dagan R, et al. Severe accident code-to-code comparison for two accident scenarios in a spent fuel pool. Ann Nucl Energy 2018;120:880–7.

[3] Karajgikar G, Nebrensky J. Reducing the user burden when running MELCOR for accident analysis for a tokamak. Fusion Eng Des 2024;202:114399.

[4] Kaiser J, Xu C, Eichler A, Garcia AS. Cheetah: Bridging the gap between machine learning and particle accelerator physics with high-speed, differentiable simulations. 2024.

[5] Moshkbar-Bakhshayesh K, Mohtashami S. Validation of codes for modeling and simulation of nuclear power plants: A review. Nucl Eng Des 2024;421:113120.

[6] Sanchez-Espinoza V, Gabrielli F, Imke U, Zhang K, Mercatali L, Huaccho G, Duran J, Campos A, Stakhanova A, Murat O, et al. KIT reactor safety research for LWRs: Research lines, numerical tools, and prospects. Nucl Eng Des 2023;414:112573.

[7] Gauntt R, Cole R, Erickson C, Gido R, Gasser R, Rodriguez S, Young M. MELCOR computer code manuals. Sandia Natl Lab NUREG/ CR 2000;6119:785.

[8] Honda T, Bartels H-W, Merrill B, Inabe T, Petti D, Moore R, Okazaki T. Analyses of loss of vacuum accident (LOVA) in ITER. Fusion Eng Des 2000;47(4):361–75.

[9] Humrickhouse PW, Merrill BJ. MELCOR accident analysis for ARIES-ACT. Fusion Sci Technol 2013;64(2):340–4.

[10] Humphries LL. EMUG MELCOR Workshop 2021. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States); 2021.

[11] Towers M, Terry JK, Kwiatkowski A, Balis JU, Cola Gd, Deleu T, Goulão M, Kallinteris A, KG A, Krimmel M, Perez-Vicente R, Pierré A, Schulhoff S, Tai JJ, Shen ATJ, Younis OG. Gymnasium. 2023.

[12] Manjavacas A, Campoy-Nieves A, Jiménez-Raboso J, Molina-Solana M, Gómez-Romero J. An experimental evaluation of deep reinforcement learning algorithms for HVAC control. Artif Intell Rev 2024;57(7):173.

[13] Shams M, Elsheikh AH. Gym-preCICE: Reinforcement learning environments for active flow control. SoftwareX 2023;23:101446.

[14] Gomes E, Pereira L, Esteves A, Morais H. PyECOM: A python tool for analyzing and simulating energy communities. SoftwareX 2023;24:101580.

[15] Cording E, Thakur J. FleetRL: Realistic reinforcement learning environments for commercial vehicle fleets. SoftwareX 2024;26:101671.

[16] Królas W, Ibarra A, Arbeiter F, Arranz F, Bernardi D, Cappelli M, Castellanos J, Dézsi T, Dzitko H, Favuzza P, et al. The IFMIF-DONES fusion oriented neutron source: evolution of the design. Nucl Fusion 2021;61(12):125002.

[17] Raffin A, Hill A, Gleave A, Kanervisto A, Ernestus M, Dormann N. Stable-Baselines3: Reliable Reinforcement Learning Implementations. J Mach Learn Res 2021;22(268):1–8.

[18] Fujimoto S, Hoof H, Meger D. Addressing function approximation error in actor-critic methods. In: International conference on machine learning. 2018, p. 1582–91.

[19] Martín-Fuertes F, García ME, Fernández P, Cortés Á, D'Ovidio G, Fernández E, Pinna T, Porfiri MT, Fischer U, Ogando F, et al. Integration of safety in IFMIF-DONES design. Safety 2019;5(4):74.

[20] D'Ovidio G, Martín-Fuertes F. Accident analysis with MELCOR-fusion code for DONES lithium loop and accelerator. Fusion Eng Des 2019;146:473–7.