# Beobench: A Toolkit for Unified Access to Building Simulations for Reinforcement Learning

Arduin Findeis
University of Cambridge
af691@cam.ac.uk

Fiodar Kazhamiaka
Stanford University
fiodar@stanford.edu

Scott Jeen
University of Cambridge, Alan Turing Institute
srj38@cam.ac.uk

Srinivasan Keshav
University of Cambridge
sk818@cam.ac.uk

## Abstract

Reinforcement learning (RL) is often considered a promising approach for controlling complex building operations. In this context, RL algorithms are typically evaluated using a testing framework that simulates building operations. To make general claims and avoid overfitting, an RL method should be evaluated on a large and diverse set of buildings. Unfortunately, due to the complexity of creating building simulations, none of the existing frameworks provide more than a handful of simulated buildings. Moreover, each framework has its own particularities, which makes it difficult to evaluate the same algorithm on multiple frameworks. To address this, we present Beobench: a Python toolkit[1] that provides unified access to building simulations from multiple frameworks using a container-based approach. We demonstrate the power of our approach with an example showing how Beobench can launch RL experiments in any supported framework with a single command.

*CCS Concepts:* • **Computing methodologies → Simulation environments**; **Reinforcement learning**.

***Keywords:*** reinforcement learning, building energy optimisation, building control, building simulation

**ACM Reference Format:**
Arduin Findeis, Fiodar Kazhamiaka, Scott Jeen, and Srinivasan Keshav. 2022. Beobench: A Toolkit for Unified Access to Building Simulations for Reinforcement Learning. In *The Thirteenth ACM International Conference on Future Energy Systems (e-Energy '22), June 28-July 1, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3538637.3538866

*e-Energy '22, June 28-July 1, 2022, Virtual Event, USA*
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9397-3/22/06.
https://doi.org/10.1145/3538637.3538866

## 1 Introduction

Reinforcement learning (RL) [18] represents a promising control method for building energy optimisation (BEO) problems. RL can work with incomplete system models, unlike classical control methods such as model predictive control, making it better suited to dynamic and complex real-world conditions. This has motivated a flurry of recent work on applying RL to BEO [5, 21, 24]. However, RL methods have been evaluated on a limited set of buildings, leaving open questions on the suitability of different RL methods for BEO *in general.*

The evaluation of RL algorithms for BEO has gone through two phases. In the first phase of work, each proposed RL algorithm was typically evaluated in its own specific building context [6, 22]. As Wölfle et al. [23] point out, this limited standardisation makes it challenging to compare the relative performance of RL methods. The focus on individual buildings could also lead to overfitting and weaken any subsequent performance claims. In the second phase of work, [2, 3, 8, 13, 16, 20], testing *frameworks* were introduced, allowing different RL algorithms to be tested on a standard set of building environments. Given the complexity of developing high-fidelity building simulations, these frameworks each provide a limited set of simulated buildings. Moreover, each framework requires the user to adapt to its unique usage and installation requirements, so that it is difficult to evaluate the same RL algorithm on multiple frameworks.

To address this research gap, we present *Beobench*, a Python toolkit that simplifies using multiple frameworks to study RL algorithms for BEO problems. We make the following contributions:

1. A Python package[1] that provides a unified interface to access multiple building control frameworks for single-agent RL algorithms.
2. Integrations of three popular building control frameworks into this package, namely BOPTEST [2, 3], Energym [16], and Sinergym [8].
3. Additional resources to simplify running experiments that apply RL for BEO, including experiment tracking and integration with RLlib [9].

[1]Open-source, available at https://github.com/rdnfn/beobench.

**Table 1.** Number of distinct building models in frameworks

| *Framework* | # Buildings | Archetypes |
|---|---|---|
| BOPTEST | 3 | Residential, Commercial |
| Energym | 6 | Residential, Commercial, Office |
| RL Testbed | 1 | Data centre |
| Sinergym | 3 | Data centre, Residential, Office |

## 2 Background and related work

In reinforcement learning, problems are formulated as a control method — or *agent* — interacting with an *environment*. In the context of building control, the environment is an interface that the agent can use to control a building. The goal is to create an agent that interacts with this building environment in a way that achieves some objective, for example minimising energy payment or emissions.

Frameworks created in the aforementioned second phase of research provide access to a set of simulated building environments. We now discuss some recent frameworks, focusing on ones that target single-agent control, are open-source, under active development[2], and accessible via a Python API. Frameworks that fit these criteria include *BOPTEST*[3] [2, 3], *Energym* [16], *Sinergym* [8], and *Reinforcement Learning Testbed for Power-Consumption Optimization* [13]. For brevity, we will refer to the latter just as *RL Testbed*. Inspired by the widely used *OpenAI Gym* framework [4], many of the aforementioned projects use the term *gym* in their name. Following this convention, we refer to the collection of the projects just mentioned as *gym frameworks*. All of the considered gym frameworks provide environments that give agents control over fully simulated buildings. The simulations are run using sophisticated software, such as *EnergyPlus* [7] or *Modelica* [11].

Developing a precise building model for these simulations is a time-consuming task and requires extensive building specification data. Thus, the aforementioned projects each use a small set of physical building models, as Table 1 shows. Note that, in addition to physical building models, each framework also provides other system variations to create a diverse set of problem formulations, such as types of controllable devices or location of weather data. A detailed discussion of environments and building models available in these gym frameworks is provided in Appendix A.

Note that there are several other frameworks that do similar tasks but do not meet our criteria, mostly because they either primarily focus on multi-agent control[4], such as *CityLearn* [20] and *GridLearn* [14], or appear to be no longer under active development, such as *Gym-Eplus* [26], *ModelicaGym* [10] and *Tropical Precooling Environment* [23]. For completeness, we mention but do not compare the frameworks *Comprehensive Building Simulator (COBS)* [25] and *EmsPy* [12] that appear to be at an early stage of development.

[2]Defined as having a git commit within the last year on the project's public repository. Given the fast pace of development within the field, packages that are not actively maintained may be at risk of becoming incompatible with popular RL libraries.

[3]BOPTEST consists of a building simulation framework [3] and a separate gym implementation [2]. For brevity, we use the term *BOPTEST* to refer to the combined framework.

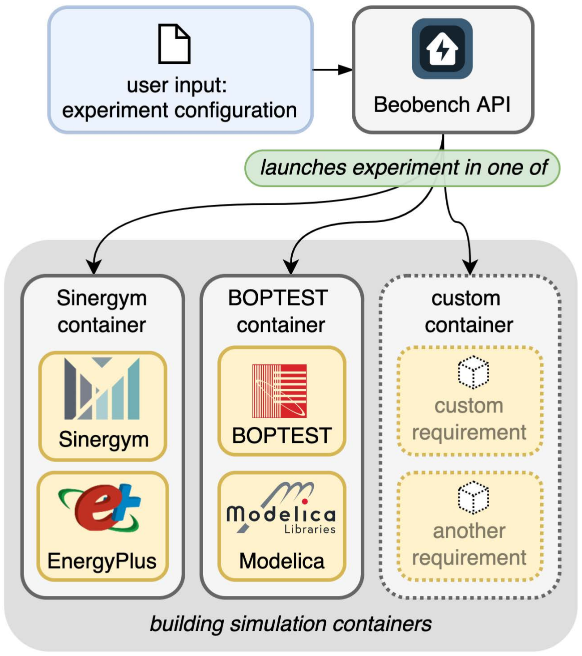

**Figure 1.** Overview of the architecture of Beobench. The user provides an experiment configuration file as input to the Beobench API, which then runs the experiments in a Docker container with all the requirements for the building simulation.

## 3 Design

Beobench is designed to extend existing gym frameworks to *support cross-framework method evaluation*. By combining buildings from multiple frameworks, Beobench provides access to the largest unified collection of building environments for single-agent RL methods. Our framework substantially reduces the effort required to evaluate an RL algorithm on this collection and obtain a robust signal on the algorithm's general applicability. For example, we can investigate whether an RL algorithm is well suited to control *houses in general*,

[4]Multi-agent RL (MARL) environments often use interfaces that differ from the *OpenAI Gym* [4] interface widely used for single-agent RL. For example, *GridLearn* [14] uses *PettingZoo* [19]. These interfaces do not necessarily easily translate to Gym and are thus beyond the current scope of the Gym-based Beobench. That said, MARL environments that are Gym-based, such as *CityLearn* [20], are straightforward to use with Beobench as a custom integration (similar to the one shown in Appendix C). In the future, such environments may be officially integrated.

**Table 2.** Features of Frameworks

| *Feature* | BOPTEST | Energym | RL Testbed | Sinergym | Beobench |
|---|---|---|---|---|---|
| Provides simulations of new buildings | ✓ | ✓ | ✓ | ✓ | – |
| Support for cross-framework method evaluation | – | – | – | – | ✓ |
| Compatibility with standard RL algorithm libraries | ✓ | – | ✓ | ✓ | ✓ |
| Support for complete problem definition | – | – | – | ✓ | ✓ |
| Support for experiment tracking | – | – | – | ✓ | ✓ |

not just a single house with its specific system represented by a single environment.

For each gym framework, potentially conflicting dependencies and system requirements need to be satisfied. Beobench provides built-in support to access and manage the requirements of multiple building control gym frameworks. Beobench manages potentially conflicting requirements for each gym framework by placing each in its own *gym container*. This container-based architecture is illustrated in Figure 1.

Note that some frameworks already provide some form of containerization. Nevertheless, a fair amount of additional tooling is necessary to make these frameworks mutually compatible. For example, to integrate BOPTEST, we had to additionally manage simulation containers within the gym container. For Energym, we added a new wrapper to make its environments comply with the OpenAI gym interface [4]. Through these types of built-in integrations with BOPTEST, Energym and Sinergym, Beobench is able to provide the largest unified collection of building environments for single-agent RL methods, giving access to all 12 physical building models of the underlying frameworks.

In addition to this core functionality, Beobench provides features that improve ease-of-use and reproducibility of experiments. Whilst some of the following features are supported by some frameworks (see Table 2), Beobench makes them available for *all frameworks* accessed via its API:

1. **Compatibility with standard RL algorithm libraries:** The OpenAI gym `Env` class[5] [4] has become the widely accepted standard interface for single-agent RL environments. Implementing this interface is a prerequisite for frameworks to be compatible with standard RL algorithm libraries, such as RLlib [9] or Stable Baselines3 [15]. Not all gym frameworks fully implement this interface. If this is the case, Beobench integrations come with a wrapper that enables compatibility of the framework with the standard RL algorithm libraries.
2. **Support for complete problem definition:** An RL building control problem has many components, including the setup of the underlying physical building simulation and the definition of RL-specific parts, such as action/control spaces and reward functions. To make RL research reproducible and comparable it is essential that there is no ambiguity about the problem definition and setup. Whilst many components can be defined through the OpenAI gym interface, this does not apply to them all. In particular, the number of steps that the agent is able to take is usually defined outside the RL environment as part of the method/training implementation. Since Beobench is able to control the full experiment stack, including the method implementation, it supports experiment definition files that unambiguously define the complete problem setup. By default, Beobench leverages RLlib [9] to facilitate the definition of components for training/method. This functionality can also be extended to work with other RL algorithm libraries.
3. **Support for experiment tracking:** An integrated experiment tracking solution can capture all relevant information about the complex building control problem space. Leveraging RLlib, Beobench is able to support popular experiment tracking tools such as *Weights and Biases*[6] and *MLflow*[7].

[5]Defined in https://github.com/openai/gym/blob/master/gym/core.py (last accessed: 26 May 2022).

Table 2 shows the support of existing frameworks for the aforementioned features, and illustrates how Beobench compares. Note that the goal of Beobench is not to provide simulations of any new buildings, but rather to enable the usage of all available buildings across the many existing frameworks.

## 4 Usage

This section describes the usage for the latest version of Beobench available at the time of publication[8].

### 4.1 General usage

Beobench is designed to be easy to both install and use. Provided that *Docker*[9] is already installed, the Beobench package can be installed simply using the command `pip install beobench`.

[6]See https://wandb.ai (last accessed: 26 May 2022).
[7]See https://mlflow.org (last accessed: 26 May 2022).
[8]*v0.5.0.* Beobench is under active development – new features may be added and usage details may change. Refer to the latest version of the online documentation at https://beobench.readthedocs.io for the most up-to-date and extensive usage guides.
[9]See https://docs.docker.com/get-docker/ (last accessed: 26 May 2022).

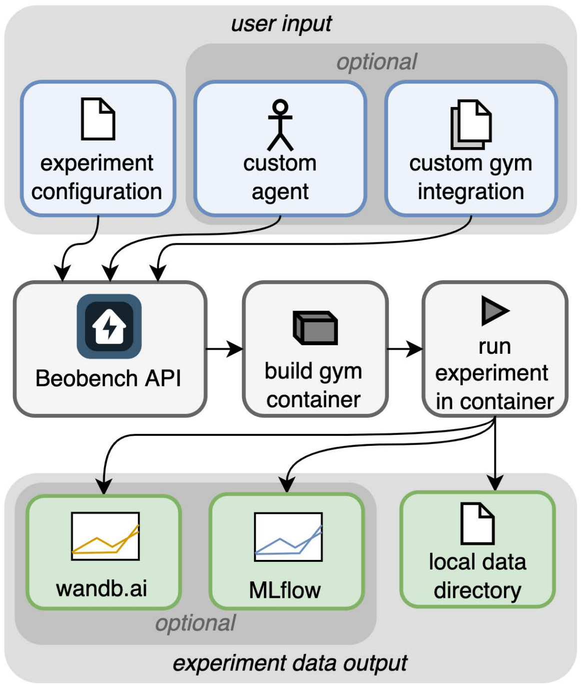

**Figure 2.** Experiment workflow with Beobench.

Once installed, Beobench uses a container-based architecture to manage the different frameworks and their system requirements. Figure 2 illustrates the workflow. To start an experiment, the user creates an *experiment configuration*, in which they can set any method from the extensive RLlib collection of RL algorithms. The user can then apply this method to any environment from the integrated frameworks BOPTEST, Sinergym or Energym. If the user wants to use a method or environment not already supported, they can optionally create a *custom agent* or a *custom gym integration* and pass it as input to the Beobench API, allowing the user to fully customize Beobench experiments to their requirements. This API can be accessed via Python or via a command line interface (CLI).

Given an experiment configuration, Beobench first builds a Docker container image custom to the gym framework of the environment used in the experiment[10]. It then starts the container and launches experiments inside it. Data from these experiments is saved in a local directory and optionally shared with experiment-tracking systems[11] such as *Weights and Biases* or *MLflow*.

## 4.2 Minimal example

Beobench can launch experiments using environments from any supported framework in just a single command. Consider the example in Figure 3: with just three commands

[10]This step is done from cache if multiple experiments use the same image – effectively skipping the step in that case.

[11]This is not supported for custom agents that do not use RLlib.

```
$ beobench run \
  --method ppo --gym sinergym \
  --env Eplus-5Zone-hot-continuous-v1

$ beobench run \
  --method ppo --gym boptest \
  --env bestest_hydronic_heat_pump

$ beobench run \
  --method ppo --gym energym \
  --env MixedUseFanFCU-v0
```

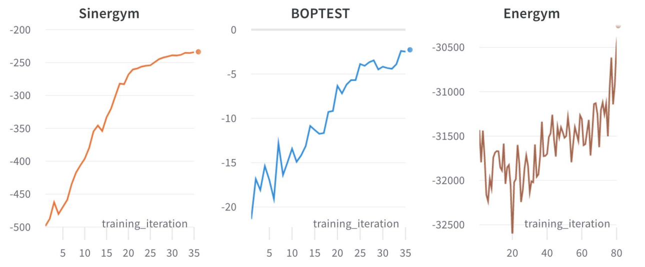

**Figure 3.** Commands to launch experiments in three different frameworks with Beobench. The plots illustrate the resulting experiments, which apply the RL method *proximal policy optimisation* (PPO) [17] to an environment from each framework. The mean episode reward over training iterations is shown.

the user is able to evaluate the same RL method, *proximal policy optimisation* (PPO) [17], in multiple frameworks. First, the method is evaluated on the *bestest_hydronic* environment[12] from BOPTEST, then on the *Eplus-5Zone-hot-continuous-v1* environment[13] from Sinergym, and finally on the *MixedUseFanFCU-v0* environment[14] from Energym. Building-specific environment information for frameworks integrated into Beobench is documented online[15]. Without Beobench, running this set of experiments would require a large amount of additional work to ensure compatibility, such as satisfying installation requirements and adapting the scripts to each framework's usage requirements.

## 4.3 Adding a gym framework

Beobench is designed to be extendable and allows the user to add new gym frameworks. Gym frameworks are integrated using *Docker build contexts* — a set of files that define a container image. In the simplest case, the build context contains two parts: a `Dockerfile` that defines a container image

[12]See https://github.com/ibpsa/project1-boptest/blob/master/testcases/bestest_hydronic_heat_pump/doc/index.html (last accessed: 26 May 2022).

[13]See https://jajimer.github.io/sinergym/compilation/html/pages/environments.html (last accessed: 26 May 2022).

[14]See https://bsl546.github.io/energym-pages/sources/mixeduse.html (last accessed: 26 May 2022).

[15]See https://beobench.readthedocs.io/en/latest/envs.html

complying with all the installation requirements of the gym framework, and an `env_creator.py` python module that defines a `create_env()` function. Given environment parameters, this function returns an environment instance that is then used by Beobench for its experiments. Some frameworks require additional tooling, for example the BOPTEST integration also has to manage separate building simulation containers. This additional tooling can be added to the build context in the same way as the other files. With this flexible integration setup, Beobench can integrate diverse gym frameworks, despite varying installation and usage requirements.

## 5 Conclusion

We have presented Beobench: a Python package that simplifies and unifies RL experiments across diverse building control frameworks. By making environments from BOPTEST, Sinergym and Energym available through a single API, the package provides the largest unified collection of building environments for single-agent RL methods. We hope that our tool will help researchers by making these standardised gym frameworks more accessible and thereby facilitate the comparison of research in this field.

Further, we anticipate that Beobench can provide the foundation for further work on benchmarking RL methods for BEO, thereby helping answer open questions about the suitability of RL methods for building control in general. We are currently working on extending Beobench by adding more pre-defined experiments, better evaluation methods [1], extensions to Docker alternatives, and more frameworks that tackle related BEO problems, such as CityLearn [20].

## Acknowledgments

We would like to thank Gilmar Findeis for his valuable advice on Docker systems and his technical support for the experimental infrastructure. Further, we would like to thank the reviewers for taking the time to read this paper in depth and for sharing their helpful comments. Arduin Findeis was supported by a University of Cambridge School of Physical Sciences Award and by the UKRI Centre for Doctoral Training in Application of Artificial Intelligence to the study of Environmental Risks (reference EP/S022961/1). Scott Jeen was funded by EPSRC, UK and Emerson Electric.

## References

[1] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville, and Marc Bellemare. 2021. Deep Reinforcement Learning at the Edge of the Statistical Precipice. *Advances in Neural Information Processing Systems* 34 (2021), 29304–29320.

[2] Javier Arroyo, Carlo Manna, Fred Spiessens, and Lieve Helsen. 2021. An OpenAI-Gym Environment for the Building Optimization Testing (BOPTEST) Framework. In *Proceedings of the 17th IBPSA Conference*. IBPSA, Bruges, Belgium. https://doi.org/10.26868/25222708.2021.30380

[3] David Blum, Javier Arroyo, Sen Huang, Ján Drgoňa, Filip Jorissen, Harald Taxt Walnum, Yan Chen, Kyle Benne, Draguna Vrabie, Michael Wetter, and Lieve Helsen. 2021. Building Optimization Testing Framework (BOPTEST) for Simulation-Based Benchmarking of Control Strategies in Buildings. *Journal of Building Performance Simulation* 14, 5 (Sept. 2021), 586–610. https://doi.org/10.1080/19401493.2021.1986574

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. *arXiv preprint arXiv:1606.01540* (2016). arXiv:1606.01540

[5] Di Cao, Weihao Hu, Junbo Zhao, Guozhou Zhang, Bin Zhang, Zhou Liu, Zhe Chen, and Frede Blaabjerg. 2020. Reinforcement Learning and Its Applications in Modern Power and Energy Systems: A Review. *Journal of Modern Power Systems and Clean Energy* 8, 6 (Nov. 2020), 1029–1042. https://doi.org/10.35833/MPCE.2020.000552

[6] Yujiao Chen, Leslie K. Norford, Holly W. Samuelson, and Ali Malkawi. 2018. Optimal control of HVAC and window systems for natural ventilation through reinforcement learning. *Energy and Buildings* 169 (2018), 195 – 205. https://doi.org/10.1016/j.enbuild.2018.03.051

[7] Drury B. Crawley, Linda K. Lawrie, Frederick C. Winkelmann, Walter F. Buhl, Y. Joe Huang, Curtis O. Pedersen, Richard K. Strand, Richard J. Liesen, Daniel E. Fisher, and Michael J. Witte. 2001. EnergyPlus: Creating a New-Generation Building Energy Simulation Program. *Energy and buildings* 33, 4 (2001), 319–331.

[8] Javier Jiménez-Raboso, Alejandro Campoy-Nieves, Antonio Manjavacas-Lucas, Juan Gómez-Romero, and Miguel Molina-Solana. 2021. Sinergym: A Building Simulation and Control Framework for Training Reinforcement Learning Agents. In *Proceedings of the 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. ACM, Coimbra Portugal, 319–323. https://doi.org/10.1145/3486611.3488729

[9] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. 2018. RLlib: Abstractions for Distributed Reinforcement Learning. In *International Conference on Machine Learning*. PMLR, 3053–3062.

[10] Oleh Lukianykhin and Tetiana Bogodorova. 2019. ModelicaGym: Applying Reinforcement Learning to Modelica Models. In *Proceedings of the 9th International Workshop on Equation-based Object-oriented Modeling Languages and Tools*. 27–36.

[11] Sven Erik Mattsson and Hilding Elmqvist. 1997. Modelica-An International Effort to Design the next Generation Modeling Language. *IFAC Proceedings Volumes* 30, 4 (1997), 151–155.

[12] mechyai. 2022. RL - EmsPy. https://github.com/mechyai/RL-EmsPy

[13] Takao Moriyama, Giovanni De Magistris, Michiaki Tatsubori, Tu-Hoa Pham, Asim Munawar, and Ryuki Tachibana. 2018. Reinforcement Learning Testbed for Power-Consumption Optimization. In *Asian Simulation Conference*. Springer, 45–59.

[14] Aisling Pigott, Constance Crozier, Kyri Baker, and Zoltan Nagy. 2021. GridLearn: Multiagent Reinforcement Learning for Grid-Aware Building Energy Management. *arXiv:2110.06396 [cs]* (Oct. 2021). arXiv:2110.06396 [cs]

[15] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research* (2021).

[16] Paul Scharnhorst, Baptiste Schubnel, Carlos Fernández Bandera, Jaume Salom, Paolo Taddeo, Max Boegli, Tomasz Gorecki, Yves Stauffer, Antonis Peppas, and Chrysa Politi. 2021. Energym: A Building Model Library for Controller Benchmarking. *Applied Sciences* 11, 8 (2021), 3518.

[17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]* (Aug. 2017). arXiv:1707.06347 [cs]

[18] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction.* MIT press.

[19] J. Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S. Santos, Clemens Dieffendahl, Caroline Horsch, and Rodrigo Perez-Vicente. 2021. Pettingzoo: Gym for Multi-Agent Reinforcement Learning. *Advances in Neural Information Processing Systems* 34 (2021), 15032–15043.

[20] Jose R. Vazquez-Canteli, Sourav Dey, Gregor Henze, and Zoltan Nagy. 2020. CityLearn: Standardizing Research in Multi-Agent Reinforcement Learning for Demand Response and Urban Energy Management. *arXiv:2012.10504 [cs]* (Dec. 2020). arXiv:2012.10504 [cs]

[21] José R Vázquez-Canteli and Zoltán Nagy. 2019. Reinforcement learning for demand response: A review of algorithms and modeling techniques. *Applied energy* 235 (2019), 1072–1089.

[22] Tianshu Wei, Yanzhi Wang, and Qi Zhu. 2017. Deep Reinforcement Learning for Building HVAC Control. In *Proceedings of the 54th Annual Design Automation Conference 2017* (Austin, TX, USA) *(DAC '17)*. Article 22, 6 pages. https://doi.org/10.1145/3061639.3062224

[23] David Wölfle, Arun Vishwanath, and Hartmut Schmeck. 2020. A Guide for the Design of Benchmark Environments for Building Energy Optimization. In *Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. ACM, Virtual Event Japan, 220–229. https://doi.org/10.1145/3408308.3427614

[24] Liang Yu, Shuqi Qin, Meng Zhang, Chao Shen, Tao Jiang, and Xiaohong Guan. 2021. A Review of Deep Reinforcement Learning for Smart Building Energy Management. *IEEE Internet of Things Journal* (2021).

[25] Tianyu Zhang and Omid Ardakanian. 2020. COBS: COmprehensive Building Simulator. In *Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. ACM, Virtual Event Japan, 314–315. https://doi.org/10.1145/3408308.3431119

[26] Zhiang Zhang and Khee Poh Lam. 2018. Practical Implementation and Evaluation of Deep Reinforcement Learning Control for a Radiant Heating System. In *BuildSys 2018 - Proceedings of the 5th Conference on Systems for Built Environments*. Association for Computing Machinery, Inc, 148–157. https://doi.org/10.1145/3276774.3276775

## A Problem variations in building control gym frameworks

This appendix provides additional information about existing building control gym frameworks that goes beyond the scope of the paper. Table 1 in Section 2 describes the number of building simulations available to the user in the existing BEO frameworks. We define this to be the number of *distinct* building envelopes one can run experiments within, rather than the number of accessible experimental *testcases*. This distinction materialises most clearly when evaluating Energym. Here, the user has access to 12 environments, however underlying these environments are only 6 building envelopes. For example, 4 testcases relate to the *Apartments* building envelope, where each has a differing state-action space and control objective. Whilst it is not the only relevant factor, we propose that differing envelopes are highly relevant for testing RL algorithm generalisability in the context of BEO. Note that Table 1 merely serves to illustrate the limited number of underlying physical building models and should *not* be considered a ranking of the frameworks. The counts in Table 1 were created as follows:

1. **BOPTEST:** We were able to count **3** distinct building models for BOPTEST:
   a. *BESTEST Case 900 room model* (used by testcases `bestest_air`, `bestest_hydronic`, `bestest_hydronic_heat_pump`)
   b. *Multi-zone residential hydronic model* (used by testcase `multizone_residential_hydronic`)
   c. *Single-zone commercial building model* (use by testcase `singlezone_commercial_hydronic`)

   Data for BOPTEST was taken from the BOPTEST repository[16]. There are 5 "testcases" based on 3 Buildings as listed above. One testcase with 5 weather options, four testcases with three weather options, all buildings have 3 pricing schemes. There are three variations of the BESTEST Case 900 room model which differ in devices available to control or overall scale. We do not include `testcase1`, `testcase2`, and `testcase3` as they are labelled as prototypes and therfore do not seem to be intended for other work.
2. **Sinergym:** We were able to count **3** distinct building models for Sinergym:
   a. *5ZoneAutoDXVAV* (used by 12 testcases)
   b. *2ZoneDataCenterHVAC_wEconomizer* (used by 4 testcases)
   c. *IWMullion* (used by 4 testcases)

   Data for Sinergym was taken from the sinergym documentation[17]. There are three buildings (in the form of `.idf` files). All buildings are varied in terms of stochasticity. One building is varied in terms of location as well. Additionally the webpage distinguishes between different action spaces — for better comparability these variations were not considered here as they are handled differently in other frameworks (e.g. not as separate environments but rather as wrappers of environments).
3. **Energym:** We were able to count **6** distinct building models for Energym (data was taken from the Energym documentation[18]):
   a. *Apartments* (used by the `ApartmentsThermal`, `ApartmentsGrid`, `Apartments2Thermal`, and `Apartments2Grid` testcases)
   b. *Offices* (used by the `Offices` testcase)
   c. *Mixed-Use* (used by the `MixedUse`)
   d. *Seminarcenter* (used by the `SeminarcenterThermostat`, and `SeminarcenterFull` testcases)
   e. *SimpleHouse* (used by the `SimpleHouseRad`, and `SimpleHouseRSla` testcases)
   f. *SwissHouse* (used by the `SwissHouseRSIa`, and `SwissHouseRSIaTank` testcases)

[16] https://github.com/ibpsa/project1-boptest/tree/master/testcases (last accessed: 26 May 2022)
[17] https://jajimer.github.io/sinergym/compilation/html/pages/environments.html (last accessed: 26 May 2022)
[18] https://bsl546.github.io/energym-pages/sources/base.html (last accessed: 26 May 2022)

Energym extensively varies the electrical devices within the buildings, and whether they are controllable.

4. **RL Testbed for EnergyPlus:** We were able to count **1** distinct building model for RL Testbed for EnergyPlus:
   a. *2ZoneDataCenterHVAC_wEconomizer_Temp_Fan* (used by 1 testcase)

   Data for this framework was taken from https://github.com/IBM/rl-testbed-for-energyplus (last accessed: 26 May 2022). The framework appears to only provide a data centre model.

## B Further application example

Note that Beobench is under active development – usage details may change. Refer to the latest version of the online documentation at https://beobench.readthedocs.io for the most up-to-date and extensive usage guides. At the time of publication, the latest version of Beobench is *v0.5.0.*

### B.1 Configuration

To get started with an experiment, we set up an *experiment configuration*. Experiment configurations can be given as a yaml file or a Python dictionary. Such a configuration fully defines an experiment, configuring everything from the RL agent to the environment and its wrappers.

Let's look at a concrete example. Consider this `config.yaml` file:

```
agent:
  origin: ./agent.py
  config:
    num_steps: 100
env:
  gym: sinergym
  config:
    name: Eplus-5Zone-hot-continuous-v1
    normalize: True
general:
  local_dir: ./beobench_results
```

Here, the first `agent` part of the configuration determines what code is run inside the experiment container. Simply put, we can think of Beobench as a tool to (1) build a special Docker container and then (2) execute some code inside that container. The code run in step (2) is referred to as the *agent script*. In the `config.yaml` file above, this agent script is set to `./agent.py` via the `agent.origin` configuration.

Before looking more closely at an `agent.py` file, let us first consider the remaining configuration. The `env` part sets the environment to `Eplus-5Zone-hot-continuous-v1` from Sinergym. The `env.config.normalize` setting ensures that the observations returned by the environment are normalized. Finally, the `general.local_dir` setting determines that all data from the experiment will be saved to the `./beobench_results` directory.

### B.2 Agent script

Next, let us have a look at an example *agent script*, `agent.py`:

```
from beobench.experiment.provider import (
    create_env,
    config
)

# create environment and get starting observation
env = create_env()
observation = env.reset()
n_steps = config["agent"]["config"]["num_steps"]

for _ in range(n_steps):
  # sample random action
  action = env.action_space.sample()
  # take selected action in environment
 observation, reward, done, info = env.step(action)

env.close()
```

The most important part of this script is the first line: we import the `create_env` function and the `config` dictionary from `beobench.experiment.provider`. These two imports are only available inside an experiment container. The `create_env` function allows us to create the environment as definded in our configuration. The `config` dictionary gives us access to the full experiment configuration (as defined before).

Note that we can use these two imports *regardless* of the gym framework we are using. This invariability allows us to create agent scripts that work across frameworks.

After the imports, the `agent.py` script above sets up a loop that takes random actions in the environment. The agent script can be considered as a template that could be customised to other requirements.

Alternatively, there are also a number of pre-defined agent scripts available, including a script for using RLlib.

### B.3 Execution

Given the configuration and agent script above, we can run the experiment using the command:

```
beobench run --config config.yaml
```

This will command will:

1. Build an experiment container with Sinergym installed.
2. Execute `agent.py` inside that container.

## C Adding a new framework/environment to Beobench

In this appendix we further discuss the process of adding a new framework or environment to Beobench. The amount of work required to do this integration differs drastically based on two factors: (1) whether the framework/environment already implements the *OpenAI Gym* [20] interface, and (2) whether there is a ready-to-use Dockerfile or Docker image available.

Out of the already supported integrations, Sinergym [8] was the easiest to add. To set up a minimal sinergym integration, we only need to create the following two files in the same folder, e.g. `sinergym_minimal/`:

1. A file named `Dockerfile` with only one line:
```
FROM alejandrocn7/sinergym:v1.7.0
```
2. An `env_creater.py` Python script defining a `create_env()` function that (when run inside the container) returns a Sinergym environment. For Sinergym, this can be achieved with the following file:
```
import gym
import sinergym

def create_env(env_config):
    return gym.make(env_config["name"])
```

To test this integration we create the following `test.yml` configuration file:

```
env:
  gym: ./sinergym_minimal/
  config:
    name: Eplus-5Zone-hot-continuous-v1
agent:
  origin: random_action
  config:
    stop:
      timesteps_total: 10
```

Given the three files described above, we can run a simple experiment using the command:

```
> beobench run -c sinergym_minimal/test.yml
```

This will launch an experiment where an agent takes ten random actions in a Sinergym environment via the integration just created.

The full Sinergym integration built into Beobench is slightly more complex as it also supports Sinergym-specific environment wrappers and customizes the Sinergym dockerfile to improve compatibility.

## D Authors' response to reviewers' comments

We thank the reviewers for their helpful feedback. There were seven overarching areas where reviewers suggested improvements. In this appendix we sort the comments by area and share our responses.

### D.1 Connection to multi-agent RL (MARL)

- Reviewer A: *The novelty could be better explained. For example, I don't see why it is a drawback of existing work that it focuses on multi-agent control, single-agent control is a special case of multi-agent control.*
- Reviewer C: *RL algorithms are more often used in multi-agent scenarios (e.g., CityLearn....). It would be very helpful if the authors can discuss on their potential strengthen and shortcoming on this. Would that require any change to the current design or implementation?*

Response: we added a footnote in Section 2 to explain that, in principle, Beobench can also be used with MARL environments as long as they are following the OpenAI Gym interface. This notably also includes the CityLearn framework. We would not say that focusing on MARL is a general drawback of existing work, but instead that limited standardisation of MARL environments makes it more difficult to provide unified access via Beobench. Thus, we decided to focus on single-agent environments in the initial phase of development. An extension to a broader set of non-gym MARL environments is a possible future extension.

### D.2 Git commit

- Reviewer A: *Similarly, how is it a drawback of existing work that the have not had a git commit in the last year, maybe they are just very well established already.*

Response: we extended the text in the relevant footnote in Section 2 to include an further explanation: "Given the fast pace of development within the field, packages that are not actively maintained are at risk of becoming incompatible with popular RL libraries."

### D.3 Application example

- Reviewer A: *It would also be useful to have some concrete application examples.*

Response: we added an additional concrete application example in Appendix B. This example shows a complete experiment setup and illustrates how Beobench can be configured via configuration files that go beyond the simple command line interface. The online documentation[19] contains additional examples and will be maintained as the package evolves.

### D.4 Amount of work required for integrations

- Reviewer B: *I would like to see some evaluation or description of how much work (lines of code, development time, etc) was required to adapt existing RL and simulation engines to the proposed package. Was this an extensive effort? Could any code be reused when interfacing with one simulation vs another? Including this information would give the reader a better impression of how much work would be required to incorporate future RL engines.*
- Reviewer C: *The authors argue that they will allow the user to fully customize SomePackage experiments to their*

[19]See https://beobench.readthedocs.io

*requirements. It would be great if the authors can add the clarification of the implementation of customized container function.*

Response: we designed the Beobench package to be easily extendable, both with respect to gym frameworks (e.g. Sinergym) and RL algorithm libraries (e.g. RLlib).

To give more details about the work required to integrate a gym (simulation) framework, we added a complete example in Appendix C. This example shows how to create and test a minimal Sinergym integration by writing (or copying) three files, each with less than ten lines of code. We hope this example helps clarify the minimum amount of work that will be required to add a new framework integration (customized container function). This basic structure can be used as a template for integrating other frameworks. In practice, some frameworks required a fair amount of additional code to be integrated into Beobench, for example to create a Gym compatibility layer or manage the framework-specific dependencies. A look at the code of the official integrations[20] shows that the BOPTEST and Energym integrations each required several hundred of lines of Python code compared to the more lightweight Sinergym integration which has less than a hundred lines of Python code. The fact that some frameworks require a fair amount of code to be compatible with each other motivates an open-source tool like Beobench where this effort only needs to be done once.

In terms of adding RL algorithms libraries, we hope that the newly added *full application example* in Appendix B demonstrates how much work may be required. In particular, the agent script in Section B.2 can be seen as a template to which custom RL algorithms can be added.

We are actively working on making both gym frameworks and RL algorithms easier to integrate.

## D.5 Multi-building benefits

- Reviewer B: *I also would have liked to see some discussion of what kind of conclusions can be drawn by being able to run a single RL algorithm over many buildings.*

Response: following this advice, we added further discussion on this topic at the start of Section 3. In addition to the points mentioned there, we believe that having access to multiple buildings via a unified interface could lay the foundation for further work on a single trained agent that is able to be applied to more than a single home.

## D.6 Access to building characteristics

- Reviewer B: *Are the parameters / characteristics of the buildings exposed in some uniform way so that the user can investigate the performance of the algorithm with the details of the target environment?*

Response: we believe that access to this kind of information is essential for users of our package and therefore have created the *Environments* section in our online documentation[21]. This section aims to give the user easy access to this information for each building environment. The section is currently work in progress, and we are actively working to keep improving and updating this section as gym integrations are added to Beobench and the environments provided by different frameworks change.

## D.7 Performance metrics

- Reviewer C: *In addition to launch time, are there any other metrics (e.g., memory size....) that could be used for evaluating the performance of SomePackage? Is SomePackage a heavy load Docker-based application?*

Response: possibly one of the most important metrics is the overall experiment time. As Beobench (SomePackage) wraps around existing gym frameworks, it inevitably introduces some computational overhead. To make Beobench a viable alternative to other installation methods, it is important to minimize this overhead. The computational overhead can be split into three parts: (1) configuration parsing, (2) container management, and environment steps (3). We are currently actively working towards reducing the computational cost associated with all three parts.

Beyond experiment time, memory and disk usage are also likely to have a significant impact on user experience. The former can be mostly controlled via API arguments, although there is a trade-off between available memory and experiment time. In terms of disk usage, the use of Docker can produce a large amount of cached images, but appropriate disk management can reduce the disk usage to a manageable level. We are considering automating this management to make Beobench more user-friendly.

Beobench is under active development, and it is difficult to provide any definite metrics at this point as they are subject to change.

## D.8 Docker alternatives

- Reviewer C: *It seems like the design can be extended to other Docker alternatives. Any thoughts on that for the non-docker users to benefit broader building research communities?*

Response: the support of Docker alternatives is something we are actively considering to integrate because the usage of Docker, whilst convenient for many users, may provide license or availability issues for some user groups, e.g. in high-performance computing (HPC) settings. We added a note on this in the Conclusion section.

[20]See https://github.com/rdnfn/beobench_contrib/tree/main/gyms

[21]See https://beobench.readthedocs.io/en/latest/envs.html