eplusr: A framework for integrating building energy simulation and data-driven analytics

Hongyuan Jia[a], Adrian Chong[b,*]

[a] *SinBerBEST Program, Berkeley Education Alliance for Research in Singapore, Singapore, 138602, Singapore*
[b] *Department of Building, School of Design and Environment, National University of Singapore, 4 Architecture Drive, Singapore, 117566, Singapore*

Abstract

Building energy simulation (BES) has been widely adopted for the investigation of building environmental and energy performance for different design and retrofit alternatives. Data-driven analytics is vital for interpreting and analyzing BES results to reveal trends and provide useful insights. However, seamless integration between BES and data-driven analytics current does not exist. This paper presents eplusr, an R package for conducting data-driven analytics with EnergyPlus. The R package is cross-platform and distributed using CRAN (The Comprehensive R Archive Network). With a data-centric design philosophy, the proposed framework focuses on better and more seamless integration between BES and data-driven analytics. It provides structured inputs/outputs format that can be easily piped into data analytics workflows. The framework also provides an infrastructure to bring portable and reusable computational environment for building energy modeling to facilitate reproducibility research.

Highlights

1. Developed an R package that integrates EnergyPlus with data-driven analytics
2. Structured inputs/outputs format that can be easily piped into data analytics workflows
3. Facilitates reproducible simulations through Docker
4. Enables flexible and extensible parametric simulations

1. Introduction

Building energy simulation (BES) is increasingly being used throughout the building's life-cycle for the analysis and prediction of building energy consumption, measurement and verification, carbon evaluation, and cost analysis of energy conservation measures (ECMs) [1,2]. It has played a growing role in the design and operation of low energy, high-performance buildings, and development of policies that drive the achievement of reducing energy use and greenhouse-gas emissions in the buildings sector [3].

BES offers an alternative approach that encourages customized, integrated design solutions, and the development of BES tools has been pronounced over the decades [3,4]. The core tools are the whole-building energy simulation programs that provide users with key building performance indicators such as energy use and demand, temperature, humidity, and costs [5]. However, BES, with an iterative nature inside, can produce a large amount of data. The volumes of the data have overwhelmed traditional data analysis methods such as spreadsheets and ad-hoc queries with a large number of factors to be considered [6]. Solutions in most existing software and applications have limited post-processing capacities on BES results. They are not flexible enough to enable a clear understanding and control of how the data is being

*Corresponding Author
Email addresses: hongyuan.jia@bears-berkeley.sg (Hongyuan Jia), adrian.chong@nus.edu.sg (Adrian Chong)

transformed [7,8]. According to a survey of 448 building energy management professionals in the U.S., there is a need to improve the efficacy and integration between data-driven analytics and BES, and efforts should be made to develop integrated tools that are capable of leveraging both methods [9].

As BES becomes more integral to many aspects of architecture design and decision-making processes, computational reproducibility has become increasingly important to researchers, designers and practitioners. Lack of credibility in BES results due to a lack of reproducibility is widely considered a problem by the energy modeling community [10]. Issues in simulation reproducibility are mainly caused by the absence of (1) an integrated workflow between BES and data-driven analytics and (2) a portable and reusable computational environment encapsulating essential software and applications to perform it.

To address these issues, this paper introduces a new framework for integrating BES and data-driven analytics. The framework is different from existing ones because of its data-centric design philosophy. The objectives are (1) to provide better and more seamless integration between BES engine EnergyPlus and R-programming data-driven analytics environment through a parametric simulation prototype with structured inputs and outputs format and (2) to build infrastructures for portable and reusable BES computational environment to facilitate reproducibility research in building energy domain. Section 2 describes the state of the art of related research and development. Section 3 introduces the concepts behind the framework, along with its implementation. Section 4 demonstrates the applications of the framework using a medium office building model with four examples, covering various topics, including data exploration, parametric simulation, optimization and calibration.

2. State of the art

2.1. BES software and applications

Over the decades, a wide variety of whole-building energy simulation programs have been developed [5]. In general, they can be classified into three categories [11], including:

1. Applications with integrated simulation engine (e.g. EnergyPlus [12], TRNSYS [13], DeST [14], ESP-r [15], IESVE [16], IDA ICE [17])
2. Software that based on a certain engine (e.g. eQUEST [18], DesignBuilder [19], OpenStudio [20], jEplus [21], Modelkit [22], GenOpt [23])
3. Plugins for other software enabling certain performance analysis (e.g. Ladybug & Honeybee [24,25], eppy [26], MLE+ [27,28], EpXL [29])

Some tools may fall into several categories, such as IESVE and IDA ICE which can also be treated as an interface software toolkit to its engine, and software OpenStudio and DeST which also provides plugins for other software to perform geometry creation and manipulation.

Choosing the appropriate combination of design options using BES is a complex task that requires the management of a large amount of information on the properties of design options and the simulation of their performance [30]. Parametric energy simulation is often needed to take into account the uncertainties and variability of different design variables. However, parametric analysis involves tedious file management tasks, repeated entry of model parameters, the application of design transformations and the execution of large-scale analyses [31], which can be time-consuming and error-prone. Parametric simulation task automation has been proven to be a useful way to reduce human intervention and improve the efficiency of large parametric analysis [32]. Table 1 gives a summary of the characteristics and capabilities of various BES software and plugins for this purpose.

In Table 1, some tools consist of graphical user interfaces (GUIs), while others use general-purpose scripting languages accompanied by a suite of programming features and libraries [32]. Among the tools, EnergyPlus is the most used simulation engine. This may be due to its advantage of free, open-source and cross-platform characteristics.

OpenStudio [20] is a free, open-source software toolkit designed for energy modeling and can be used to efficiently create or modify models, manage individual or multiple simulations, and visualize results. OpenStudio has its own format (`.OSM`) and schema for EnergyPlus model representation which will eventually

Table 1: Summary of characteristics and capabilities of BES software and plugins

Name	Simulation engine	Cross-platform	Free	GUI	Open-source	API language	Semantic API[1]	Supports optimization[2]	Supports calibration	BIM interoperability
IESVE [16]	IESVE			✓		Python	✓	✓	✓	✓
IDA ICE [17]	IDA ICE			✓				✓		✓
eQUEST [18]	DOE-2		✓	✓						
DesignBuilder [19]	EnergyPlus			✓		C#, Python		✓		✓
OpenStudio PAT [20]	EnergyPlus	✓	✓	✓	✓	Ruby	✓	✓	✓	✓
Ladybug & Honeybee [24]	EnergyPlus		✓	✓	✓	Python	✓			✓
jEplus [21]	EnergyPlus	✓	✓	✓	✓			✓		
Modelkit [22]	EnergyPlus		✓			Ruby				
MLE+ [27]	EnergyPlus		✓	✓	✓	Matlab		✓		
EpXL [29]	EnergyPlus		✓	✓	✓	VBA		✓		
eppy [26]	EnergyPlus	✓	✓		✓	Python				

[1] Further abstraction classes to directly perform geometry transformations, HVAC system manipulation, etc.
[2] Only built-in features are considered. So as for calibration support. Some tools can be further coupled with other software or libraries to perform optimizations

be translated into EnergyPlus models. Parametric Analysis Tools (PAT) is a GUI application that is part of the OpenStudio toolkit and is capable of utilizing customizable and shareable parametric descriptions of ECMs [33]. It leverages OpenStudio Measures which are reusable scripts written in Ruby programming language to manipulate OpenStudio models and can be used to compare manually specified combinations of measures, optimize designs, calibrate models and perform parametric sensitivity analysis. Ladybug and Honeybee are plugins developed for Rhino Grasshopper [24,25]. Ladybug is used to import and analyze Energy standard weather data (EPW) while Honeybee is used to create, run, and visualize OpenStudio and EnergyPlus simulation results. They leverage the visual programming interface provided by Grasshopper and thus are capable of performing parametric geometrical modeling. However, Honeybee is not able to access all features of OpenStudio. Both OpenStudio and Honeybee have built further abstractions that are capable of performing building geometry transformation and restructuring HVAC (Heating, Ventilation, and Air-Conditioning) systems.

Since the primary Input Data Files (IDFs) of EnergyPlus are all ASCII text-based, several tools directly update the building energy models by processing and manipulating the text files, without taking into account the complex hierarchical structure in the model components. jEplus [21] is a software written in Java programming language to perform complex parametric analysis on multiple design parameters. It allows users to describe the parameters and their values using customized symbols via a GUI and automatically create parametric models using text-substitution [34]. Modelkit [22] automates the generation and management of EnergyPlus models via its templates and scripting tools written in Ruby. These frameworks are designed for simplicity and flexibility and are mainly focusing on generating parametric models based on an existing seed model, instead of creating a new one from scratch.

For further customized automation tasks, several tools are interfacing EnergyPlus with scripting programming languages. MLE+ integrates EnergyPlus and scientific computation and design capacities of Matlab for controller design and can be used to implement and simulate advanced control algorithms of building systems [27,28]. EpXL [29] is an EnergyPlus Microsoft Excel user-interface written in Visual Basic for Applications (VBA) that enables the import and export of IDF data files, parametric analysis, and optimization. It is capable of displaying a compact tabular overview of input data and automatically importing simulation output into Excel, with a link for viewing the 3D model. eppy [26] is a library for interfacing EnergyPlus with Python programming language. It parses EnergyPlus models into a Python object and provides low-level programmatic access to EnergyPlus inputs, making it possible to leverage rich scientific computing libraries in Python.

The tools mentioned above may have overlappings in features. However, they are tailored for different purposes and use cases, with the primary focus on ease the time-consuming and error-prone process of

creating and managing parametric simulations.

2.2. Data-driven analytics of BES data

Currently, there is a growing body of scientific literature on the application of advanced mathematical algorithms for building design using BES [35]. Generally, data-driven analytics encompasses the whole data analysis process beginning with data extraction and cleaning, and extends to data analysis, description and summarization [36,37]. The processing of the simulation results forms an essential step before any application of data analytics.

However, the output of common BES tools is not always friendly in format for applying these methods, which makes data pre-processing an essential but time-consuming and laborious process for any data-driven analytics for BES data. This highlights the potential areas for improvements in data extraction and result presentation in a clear and intuitive manner for data analytics. Table 2 gives a summary of capabilities related to data processing of BES tools mentioned in Table 1.

Even most BES tools provide summary reports with various details, it is still quite common to perform post-processing and apply customized and more advanced algorithms to the simulation results. Unfortunately, most existing tools listed in Table 2 have limited capacities with this regard. Open-source programming environments such as R [38] and Python [39] are promising in providing solutions for large-scale data analytics. They have become widely-used research tools that provide access to many well-documented packages for various data mining, machine learning, and data visualization applications [36,40]. Even though a recent survey [9] has highlighted the urgent need for an integrated solution, fewer efforts have been made in terms of providing a seamless, integrated approach to bridge the gap between BES and data-driven analytics.

Table 2: Summary of capabilities related to data processing of BES software and plugins

Name	Weather data handling[1]	Further data extraction[3]	SQL-based structural output[2]	Post-processing capabilities[4]
IESVE [16]	✓	✓		★★★
IDA ICE [17]		✓		★★
eQUEST [18]		✓		★
DesignBuilder [19]		✓	✓	★★
OpenStudio PAT [20]	✓	✓	✓	★★★
Ladybug & Honeybee [24]	✓	✓		★★
jEplus [21]		✓	✓	★★★
Modelkit [22]				★
MLE+ [27]				★
EpXL [29]		✓		★★
eppy [26]		✓		★★★

[1] The capabilities of extracting and modifying data from weather files
[2] The capabilities of using SQL (Structured Query Language) queries to extract specific simulation results
[3] The capabilities of extracting further customized summary data, instead of solely based on the built-in functionalities of the simulation engine
[4] The capabilities of performing further data analyses on the extracted simulation results. The number of stars indicates the relative capabilities

2.3. Reproducible research of BES

BES involves multiple scientific processes, making its reproducibility difficult. Reproducibility is defined as the ability to recompute data analytic results, given an observed data set and knowledge of the data analysis pipeline [41]. Reproducible research has received an increasing level of attention throughout the scientific community and the public at large [42]. In a survey of 1576 researchers, more than 70% failed to reproduce another scientist's experiments, and more than 50% failed to reproduce their own experiments [43]. There was also a consensus of a significant reproducibility crisis. Currently, improving computational reproducibility has become an important step to increase the credibility in BES results [10]. The reasons

for the BES reproducible issue are two-fold: (1) missing seamless integration between simulation and data analysis workflows and (2) absence of a portable and reusable computational environment. Most users prefer to use GUI applications which makes it intuitive and easy to execute specific tasks. However, GUI tools have constraints on flexibility as the users have to specify exactly what and how features of the design can be manipulated and often are not be able to provide a good workflow for repeating that task across a broader range of situations on different systems. In this case, manual steps have to be performed using other tools, such as a spreadsheet or command-line tools, which introduces additional transcription burden and results in a non-reproducible process [31]. Sometimes, custom solutions have to be created from scratch to automate part portions of the workflows, which may lead to new inefficiencies and potential errors. Currently, no widely adopted solution is able to integrate all processes into one single platform.

Moreover, BES often involves the use of multiple applications, software and platforms. To perform crucial scientific processes such as replicating the results, extending the approach or testing the conclusions in other contexts, the indispensable step is to install the software used by the original researchers, which sometimes can become immensely time-consuming if not impossible. It is easy to underestimate the significant barriers raised by a lack of familiar, intuitive, and widely adopted tools for addressing the challenges of computational reproducibility [42].

3. Methodology

To achieve seamless integration between BES and data-driven analytics, we propose a framework consisting of the following (Fig. 1):

1. I/O processors for structuring BES inputs and outputs for seamless integration with data analytics workflow.
2. A parametric prototype for conducting flexible and extensible parametric simulations.
3. A computational environment that is based on Docker containerization [44] to facilitate reproducibility research in the energy simulation domain.

The first two components have been packaged into a free, open-source R package *eplusr*[1] which is distributed using CRAN (The Comprehensive R Archive Network). The third component has been encapsulated using Docker containerization and is distributed using Docker Hub[2].

3.1. I/O processors

The I/O processors are implemented through three modules shown in Fig. 1, including:

1. Relational Database module to represent EnergyPlus models and weathers in relational databases,
2. Object-Oriented Programming (OOP) Model API module for tidy data model modification APIs
3. Tidy Data Interface module for querying and structuring BES outputs in tidy format.

3.1.1. Relational databases

The Relational Database module is developed to read, parse and represent EnergyPlus models and weathers in relational databases. EnergyPlus Input Data File (IDF) is based on the data schema that is defined in the Input Data Dictionary (IDD). In the proposed framework, data of an IDF and the corresponding IDD are stored as Relational Databases (RD). RD was first proposed by Codd [45] and has become the dominant database model for a number of Relational Database Management Systems (RDMS). It organizes data in a set of rectangular tables with rows and columns. Each table has a primary key which is an unique identifier constructed from one or more columns. A table is linked to another by including the other table's primary key (also called a foreign key).

[1] GitHub Repository: https://github.com/hongyuanjia/eplusr
[2] Docker Hub Link: https://hub.docker.com/r/hongyuanjia/eplusr

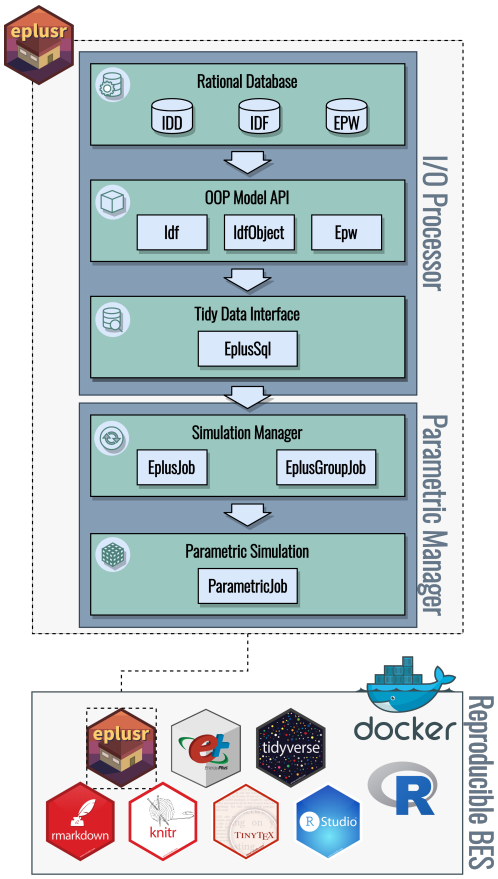

Figure 1: An architecture overview of the proposed framework which includes three main components: (1) I/O processors, (2) Parametric prototype and (3) Computational environment for reproducible BES using Docker containerization

Fig. 2 shows the structure of RD for an EnergyPlus IDF and IDD. The RD data structure follows the idea of database normalization where each variable is expressed in only one place, avoiding any data redundancy. The hierarchy structure of the IDF data schema is retained through various tables. Data integrity is maintained via relations among table variables. Each RD has a `reference` table to store the referencing relations among various field values. To modify an IDF is equal to change the corresponding fields in its RD tables. The RD structure provides the capability to quickly perform data wrangling and fast table joining among entities and variables.

3.1.2. Object-oriented programming model API

The Object-oriented programming (OOP) Model API module enables users to perform queries and modifications on EnergyPlus models programmatically. OOP [46] is a programming paradigm that focuses on the objects to manipulate rather than the logic required to manipulate them. It provides a clear modular structure for programs and is good for defining abstract data types. OOP hides implementation details and makes it possible to develop a clearly defined interface for each abstraction.

Fig. 3 gives an overview of the OOP Model API module. It introduces three groups of classes, including (1) `Idd` class and `IddObject` class for a whole and part of an IDD, (2) `Idf` class and `IdfObject` class for a whole and part of and IDF, and (3) `Epw` class for an EPW (EnergyPlus Weather). Each class provides a number of methods to manipulate the encapsulated data. An extensive rule-based data model validator has been developed to check the integrity of data before any modifications.

`Idf` class exposes flexible interfaces to modify field values in different scope levels, including single-object

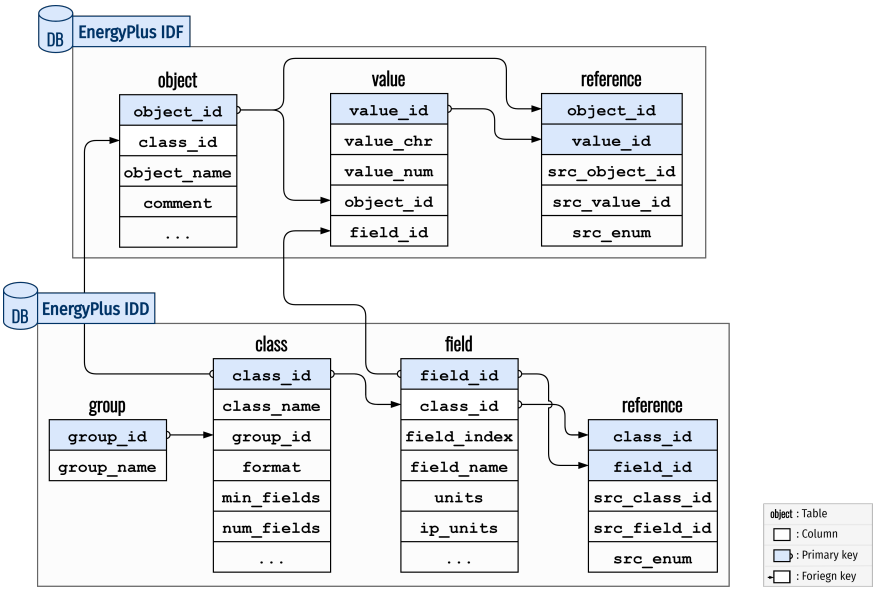

Figure 2: Structure of relational databases for an EnergyPlus IDF and IDD

level, grouped-object level, and whole-class level, enabling to alter a number of objects at the same time. Both `Idf` and `IdfObject` class provide a `to_table()` method to extract certain or all parts of a model into one `data.table` object, which is an extension of R's table representation but extremely optimized for fast computation [47]. The `load()` and `update()` methods in `Idf` class can take any model data in table format as input, create and modify large number of objects accordingly. Section 4 demonstrates some of the APIs in this module.

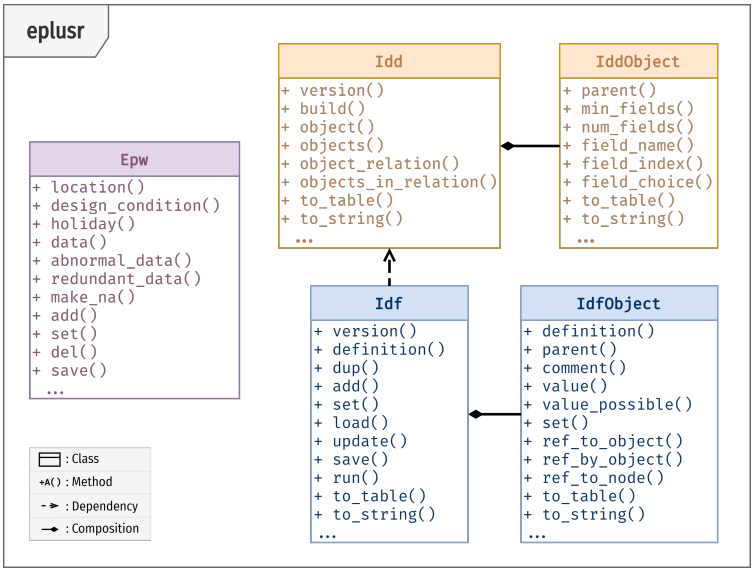

Figure 3: Overview of OOP Model API

3.1.3. *Tidy data interface*

The Tidy Data Interface module is designed to extract and represent EnergyPlus simulation results from the SQLite output into tidy tables. The concept of tidy data format was first proposed by Wickham [48], as a standard way of mapping the meaning of a dataset to its structure. It means that each variable forms a column, each observation forms a row, and each type of observational unit forms a table (see Table (b) in Fig. 4). This structure makes it intuitive for an analyst or a computer to extract needed variables. It is particularly suited for vectorized programming languages like R. The layout ensures that values of different variables from the same observation are always paired [48,49] and is well fitted for data analyses using the *tidyverse* R package ecosystem [50].

Table (a) in Fig. 4 shows an example of the standard format from EnergyPlus CSV table output, while Table (b) gives the tidy representation of the same underlying data using the tidy data interface. Although the structure of Table (a) provides efficient storage for completely crossed designs, it violates with the tidy principles, as variables form both the rows and columns and column headers are values, not variable names. Several values are concatenated in column headers, including variable `Key Value`, `Variable Name`, `Units` and `Reporting Frequency`. Additional data cleaning efforts are needed to work with this structure, especially considering the missing values (`NA` in row 2 and 4 in Table (a)) introduced by the aggregation of various reporting frequencies, which may add new inefficiencies and potential errors. In Table (b), values in column headers have been extracted and converted into separate columns, and a new variable called `Value` is used to store the concatenated data values from the previously separate columns. Moreover, instead of presenting date and time as strings in Table (a), the tidy data interface splits its components into four new variables, including `Month`, `Day`, `Hour` and `Minute`. Taken together, Table (b) forms a nine-variable tidy table and each variable matches the semantics of simulation output. Considering the times of data analysis operations to be performed on the values in a variable, the advantage of structuring values in a standard and straightforward way stands out. It can facilitate initial exploration and analysis of data and to simplify the development of data analysis tools that work well together [48].

(a)

	Date/Time	Key1: Variable1 [J] (30-min)	Key2: Variable1 [J] (Hourly)	Key1: Variable2 [W] (30-min)
			KeyValue Name Units	Frequency
1	m/d H1:00	value1	value5	value7
2	m/d H1:30	value2	NA	value8
3	m/d H2:00	value3	value6	value9
4	m/d H2:30	value4	NA	value10

Tidy Data Interface

(b)

	Month	Day	Hour	Minute	KeyValue	Name	Units	Frequency	Value
1	m	d	H1	0	Key1	Variable1	J	30-min	value1
2	m	d	H1	30	Key1	Variable1	J	30-min	value2
3	m	d	H2	0	Key1	Variable1	J	30-min	value3
4	m	d	H2	30	Key1	Variable1	J	30-min	value4
5	m	d	H1	0	Key2	Variable1	J	Hourly	value5
6	m	d	H2	0	Key2	Variable1	J	Hourly	value6
7	m	d	H1	0	Key1	Variable2	W	30-min	value7
8	m	d	H1	30	Key1	Variable2	W	30-min	value8
9	m	d	H2	0	Key1	Variable2	W	30-min	value9
10	m	d	H2	30	Key1	Variable2	W	30-min	value10

Figure 4: An example of tidy BES output data representation using Tidy data interface where Table (a) is the standard output format of EnergyPlus CSV table and Table (b) is the tidy representation of the same underlying data using the Tidy data interface

Fig. 5 shows an implementation overview of the tidy data interface for EnergyPlus variable and meter outputs using EnergyPlus SQLite output. SQLite is a mature and widely-employed RDMS [51]. The main benefit of using the EnergyPlus SQLite output format is that it contains all of the data in standard reports, variable and meter output, and also a number of input and output summaries. An `EplusSql` class is introduced with interfaces to retrieve outputs of any given time period and for any variables in a consistent

manner. It is achieved by sending SQL (Structured Query Language) queries, a domain-specific language for RDMS, to the SQLite simulation output database. The results are outcomes of joining operations on four tables, including `Time`, `EnvironmentPeriods`, `ReportDataDictionary`, and `ReportData`. However, the time components in the SQLite outputs fail to assemble complete time-series data, due to missing a year specification[3], making it impossible to directly apply time-series-based algorithms. To solve this issue, a year derivation algorithm is implemented that calculates a proper year value for each run period based on the date and time components, and compose a complete series of `POSIXct` values, which is the standard date-time class in R.

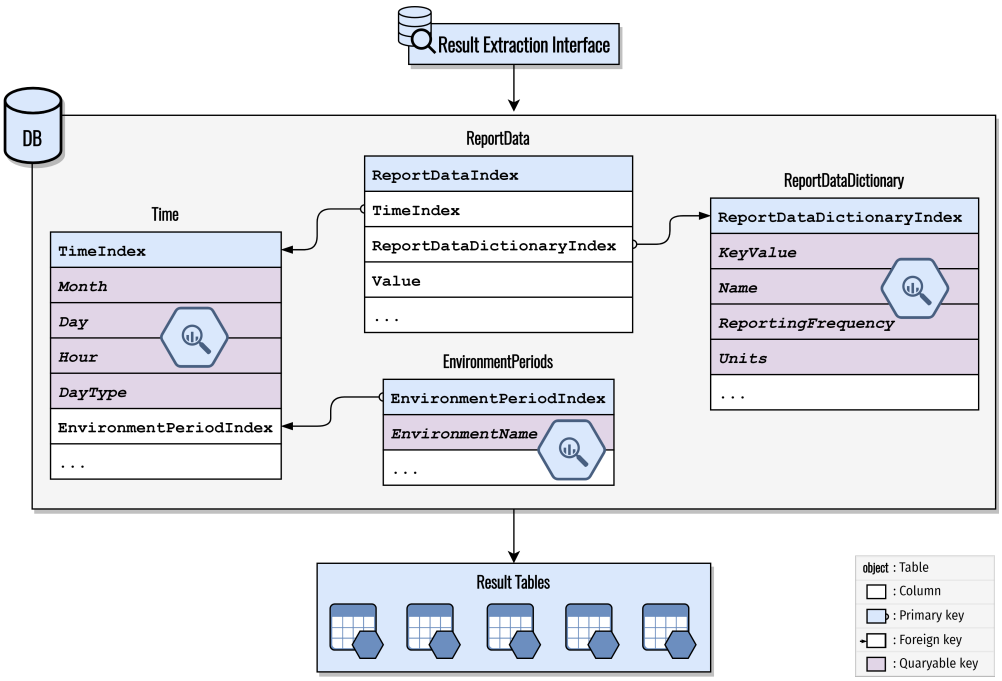

Figure 5: Overview of the tidy data interface for variable and meter outputs

3.2. Parametric prototype

The parametric prototype in the framework provides a set of abstractions to ease the process of parametric model generation, design alternative evaluation, and large parametric simulation management. An overview of the parametric prototype implementation is shown in Fig. 6.

A parametric simulation is initialized using a seed model and a weather file. Design alternatives are specified by applying a *measure* function to the seed model. The concept of *measure* in the prototype is inspired by a similar concept in OpenStudio [20] but tailored for flexibility and extensibility. A measure is simply an R function that takes an `Idf` object and any other parameters (e.g. t_1 to t_5 in Fig. 6) as input, and returns a set of modified `Idf` objects as output, making it possible to leverage other modules in the framework and apply statistical methods and libraries existing in R to generate design options. After a measure is defined, the method `apply_measure()` takes it and other parameter values specified to create a set of models. The `run()` method will run all parametric simulations in parallel and place each simulation outputs in a separate folder. All simulation metadata will keep updating during the whole time and can be retrieved using the `status()` method for further investigations.

[3] A `Year` field was added in the recent version of EnergyPlus. But old versions of EnergyPlus are still widely used.

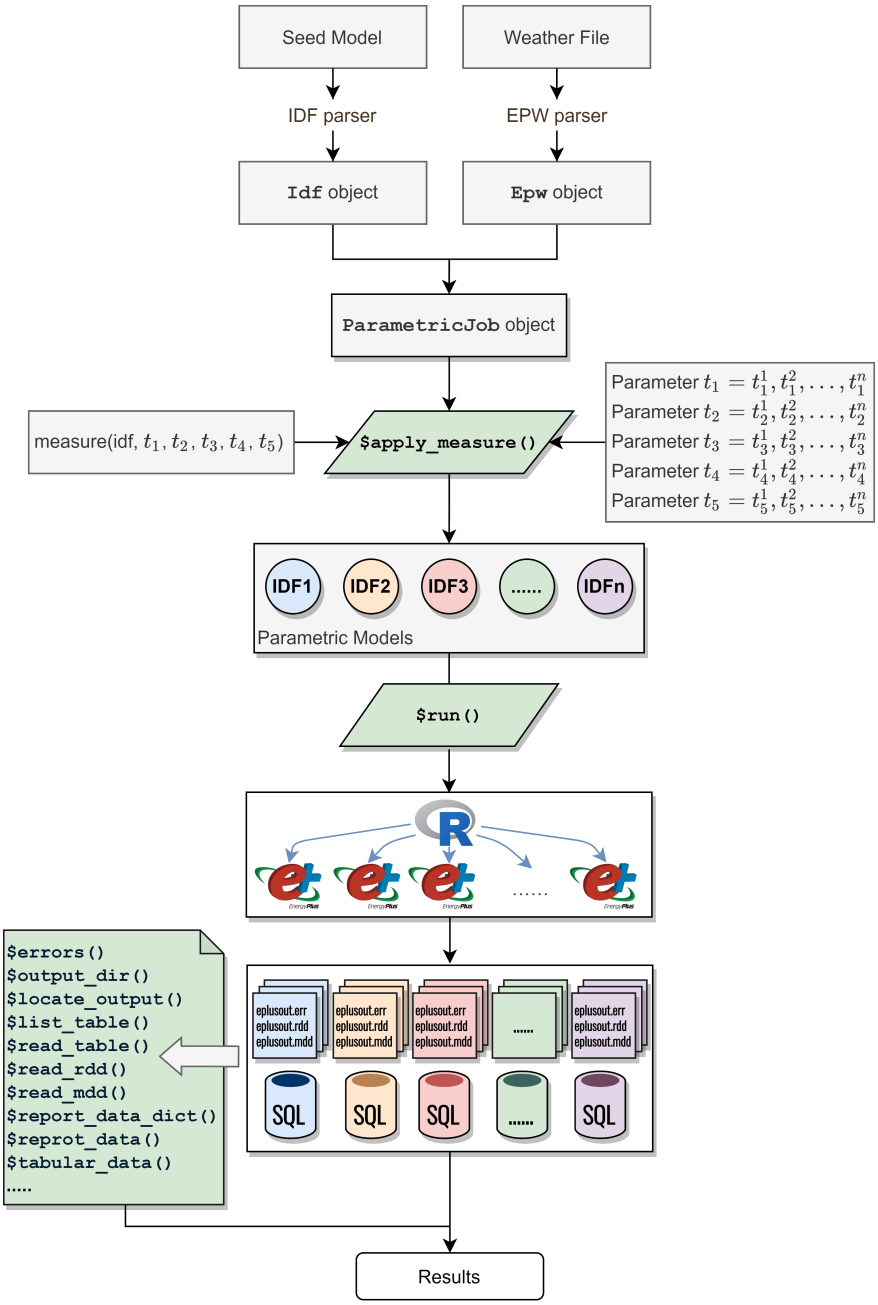

Figure 6: Workflow of a parametric simulation

The `ParametricJob` class leverages the tidy data interface to retrieve parametric simulation results in a tidy format. Despite that, a number of methods are also provided to read various output files, including simulation errors (`eplusout.err`), report data dictionary (`eplusout.rdd`), and meter data dictionary (`eplusout.mdd`). For all resulting tidy tables, an extra column containing the simulation job identifiers is prepended in each table. It can be used as an index or key for further data transformations, analyses and visualization to compare results of different simulated design options.

The proposed parametric prototype is designed to be simple yet flexible and extensible. One good

example of the extensibility of this framework is the epluspar[4] R package, which provides new classes for conducting specific parametric analyses on EnergyPlus models, including sensitivity analysis using the Morris method [52] and Bayesian calibration using the method proposed by Chong [1]. All the new classes introduced are based on the `ParametricJob` class. The main difference mainly lies in the specific statistical method used for sampling parameter values when calling `apply_measure()` method. Few examples of this application have been provided in Section 4.

3.3. Computational environment for reproducible BES

The Docker containerization for BES aims to provide infrastructures to bring portable and reusable computational environments to facilitate reproducible BES applications. Peng [41] summarized two major components to successful reproducible research: (1) data, i.e. the availability of raw data from the experiment, and (2) code, i.e. the availability of the statistical code and documentation to reproduce the results. In the context of BES, these two component will be (1) the building energy models and (2) the code to perform simulations and following data-driven analytics. However, the complex and rapidly changing nature of computer environments makes it immensely challenging to reproduce the same workflow and results even with the original data and code are available. To address this issue, a reproducible BES computational environment has been developed based on the Docker containerization technology, which captures the full software stack, including all software dependencies in a portable and reusable image.

Docker [44] is a popular open-source tool for containerization and has shown its potential to improve computational reproducibility [42,53]. The Rocker Project was launched in 2014 as a collaboration to provide high-quality Docker images containing the R environment and has seen both considerable uptakes in the R community and substantial development and evolution [54]. The proposed reproducible BES computational environment is built upon the `rocker/verse` images. It contains four groups of toolchains needed for common BES and data-driven analytics workflows using the eplusr framework:

1. Statistical computing environment, including the latest R environment and RStudio Server, a web-based integrated development environment for R programming
2. BES engine, including EnergyPlus of specified version and the eplusr R package
3. Data analytics toolkits, including a collection of tidyverse [50] R packages for data import, tidying, manipulation, visualization and programming
4. Literate programming environment, including R Markdown related packages for dynamic document generation

The first three have been described in previous sections. Literate programming is a programming paradigm introduced by Knuth [55] in which the explanation of a computer program is given, together with snippets of source code. Recently, there have been significant efforts to develop literate programming infrastructure to reproducibly perform and communicate data analyses, including R Markdown [56] , Jupyter notebook [57], just to name a few.

The R Markdown format is powered by the knitr R package [58] and Pandoc [59]. Knitr executes the computer code written in various programming languages embedded, and converts R Markdown to Markdown. Pandoc processes the resulting Markdown and render it to various output formats, including PDF, HTML, Word, etc. The R Markdown format has been a widely adopted authoring framework for data science. It can be used to both save and execute code and generate high-quality reports that can be shared with an audience. Together with *rmarkdown* [60] and *tinytex* [61] packages, the proposed BES computational environment can be easily adapted to any R-centric workflows and enables researchers in the BES field to build and archive reproducible analytics.

The source files of Docker configuration were written in several text files so-called Dockerfiles and are publicly available and hosted via GitHub[5]. Further evolutions can be taken to make the computational environment tailored to different audiences and use purposes. The docker approach is suited for moving

[4]GitHub Repository: https://github.com/hongyuanjia/epluspar
[5]GitHub Repository: https://github.com/hongyuanjia/eplusr-docker

between local and cloud platforms when a web-based integrated development environment is available, such as RStudio Server [42], providing the scalability potential for large cloud-based BES computation.

4. Applications

To show how the eplusr framework can be used, examples are presented in four topics: (1) data exploration, (2) parametric simulation, (3) multi-objective optimization (MOO) using Genetic Algorithm (GA), and (4) Bayesian calibration. For all examples, the U.S. Department of Energy (DOE) medium office reference building model in compliance with Standard ASHRAE 90.1 – 2004 [62] is used. Fig. 7 shows a 3D view of the building geometry. It is a 3-story, 15-zone medium office building with a total floor area of 4982 m^2. A central packaged air conditioning unit with a gas furnace is equipped on each story. The air distribution systems are Variable Air Volume (VAV) terminal boxes with electric reheating coils. The typical meteorological year 3 (TMY3) weather data of Chicago was used for all the simulations.

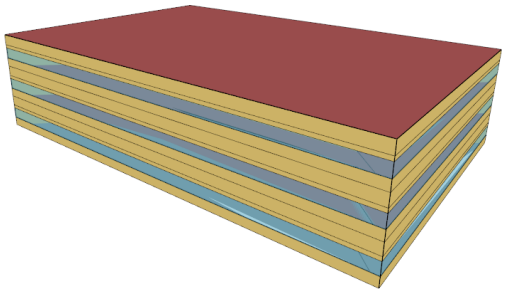

Figure 7: 3D view of DOE medium office reference building

4.1. Data exploration

Data exploration is an essential aspect of BES. It is often used to reduce large volumes of simulation data to a manageable size so that efforts can be focused on analyzing the most relevant data. This example demonstrates the data exploration process of obtaining (1) energy use intensity (EUI) and (2) heating and cooling demand profile using annual simulation results. The energy use intensity (EUI) is one key indicator for building energy performance, and its breakdown can provide potential directions of where ECMs should be applied to reduce energy usage. When evaluating the feasibility of free-cooling applications in buildings, the heating and cooling demand profile plays an essential role in the determination of the potential. This example showcases the basic features of the proposed framework with the main focus on how the tidy data interface can provide a seamless workflow to extract BES output, feed it into data analysis pipelines and turn the results into understanding and knowledge.

Fig. 8 shows an overview of a typical data exploration workflow using BES output extracted by the tidy data interface described in Section 3.1.3. Listing 1 shows the R code to achieve it.

Lines 38 – 53 in Listing 1 shows how to use methods `tabular_data()`, `read_table()` and `report_data()` provided by the tidy data interface to extract building area and building energy consumption, zone metadata, and cooling and heating demands, with all formatted in a tidy representation. Note that instead of presenting the simulated date and time as strings, the `report_data()` adds a time-series column `datetime` in `POSIXct` based on a derived year value using the algorithm described in Section 3.1. Moreover, the tidy data interface also provides a number of additional columns shown in Fig. 5, which makes it quite convenient and straightforward to directly perform further data transformations. Lines 93 – 107 in Listing 1 demonstrate the benefits of the tidy format in selecting columns using `select()`, subsetting rows using `filter()`, sorting rows using `arrange()`, adding new variables using `mutate()`, summarizing data using a combination of `group_by()` and `summarize()`, joining tables using `left_join()`, and data visualization using `ggplot()`.

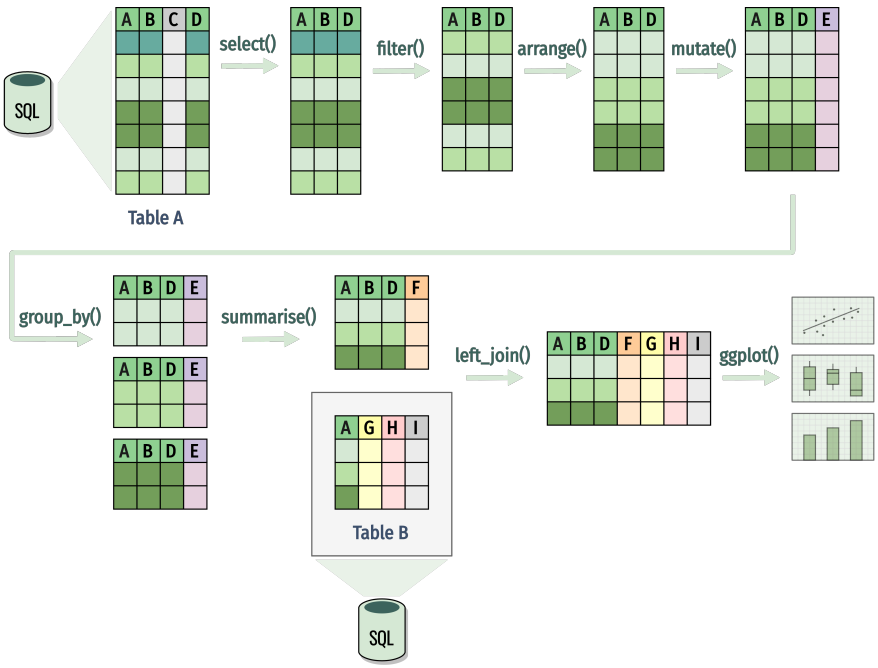

Figure 8: Workflow overview of data exploration using BES output extracted by the tidy data interface

Based on the building energy consumption data (line 39 in Listing 1) and the building area (line 36 in Listing 1), the electricity EUI breakdown from various end-use categories was calculated, and a pie chart was created (shown in Fig. 9) using only 13 lines of codes (line 56 – 77 in Listing 1). From Fig. 9, we can see that most of the energy has been consumed by interior electric equipment, followed by indoor lighting. It indicates that ECMs which help to reduce the plug loads and lighting power density (LPD) may have a promising potential in improving the overall energy performance. We will perform further investigations on this in Section 4.2.

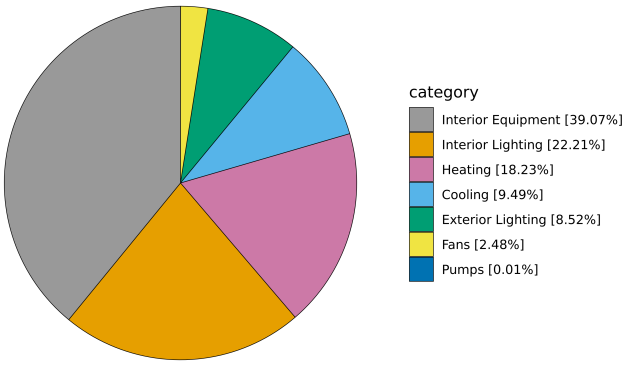

Figure 9: Annual electricity EUI breakdown

Fig. 10 shows the profile of monthly heating and cooling demands in a unit of MJ/m^2. It is calculated based on the zone metadata (line 42 in Listing 1) and hourly air system heating and cooling energy outputs (lines 46 – 50 in Listing 1). With the tidy format and additional metadata columns, these two data fit well in the data pipeline, making it straightforward and intuitive to perform data transformation and visualization. Fig. 10 is a result of only around 30 lines of code (line 41 – 50 and 79 – 119 in Listing 1). In Fig. 10, we can see that during the transition seasons, including March, April, October, and November, the heating and

cooling demands are relatively small compared to summer and winter seasons, indicating the potential of free-cooling applications.

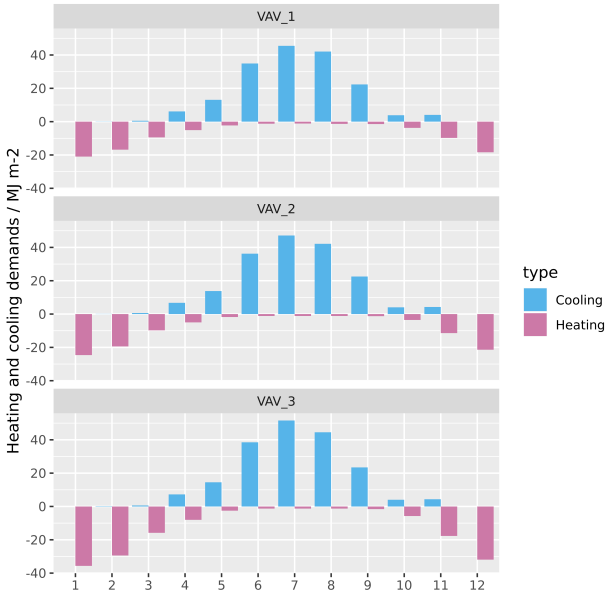

Figure 10: Monthly heating and cooling demand profile

4.2. Parametric simulation

This example demonstrates the process of performing parametric simulation analyses using the proposed parametric prototype. The main focuses are on showcasing the capabilities of (1) creating parametric models by applying measures and (2) easing the comparative analysis by reusing code snippets developed in data exploration process.

As shown in Fig. 9, the plug loads and interior lighting systems consumed more than 60% of total electricity. In this example, we will investigate the energy-saving potentials of ECMs on reducing the plug loads and LPD. Fig. 11 gives an overview of the workflow and Listing 2 shows the actual R code.

Measures are functions that describe how an energy model should be modified based on input parameter values. Lines 4 – 14 in Listing 2 shows a simple measure that modifies the LPD. The core code is line 14 that assigns all related fields in a whole class to input values, taking advantage of the flexibly-scoped OOP model API. Lines 20 – 40 in Listing 2 shows a measure that modifies the off-work schedule values of plug loads by multiplying a specified reduction faction value. It demonstrates how objects in an energy model can be translated into a table and how to use the modified table to alter corresponding object values.

Different measures can be chained together and supplied to the `apply_measure()` method to create parametric models. Each model will be tagged with a *case* name as an identifier. As demonstrated in lines 42 – 57 in Listing 2, the combined measure `ecm` is used to create six models with various combinations of LPD and plug loads control strategies.

After calling the `run()` method to conduct parallel runs of simulations (line 67 in Listing 2), the tidy data interface can be used to extract any simulation outputs of interest from the SQL database using `report_data()`, `tabular_data()`, etc. In this example, the building energy consumptions of all six models are extracted using one line of code (line 67 in Listing 2). The resulting data format is the same as that of a single simulation and is equivalent to bind rows from six tables into one tidy table. A `case` column is prepended using the names specified in line 56 in Listing 2. It works as an identifier to group the results by different parametric models using `group_by()` and `nest()` functions from the tidyverse package. This

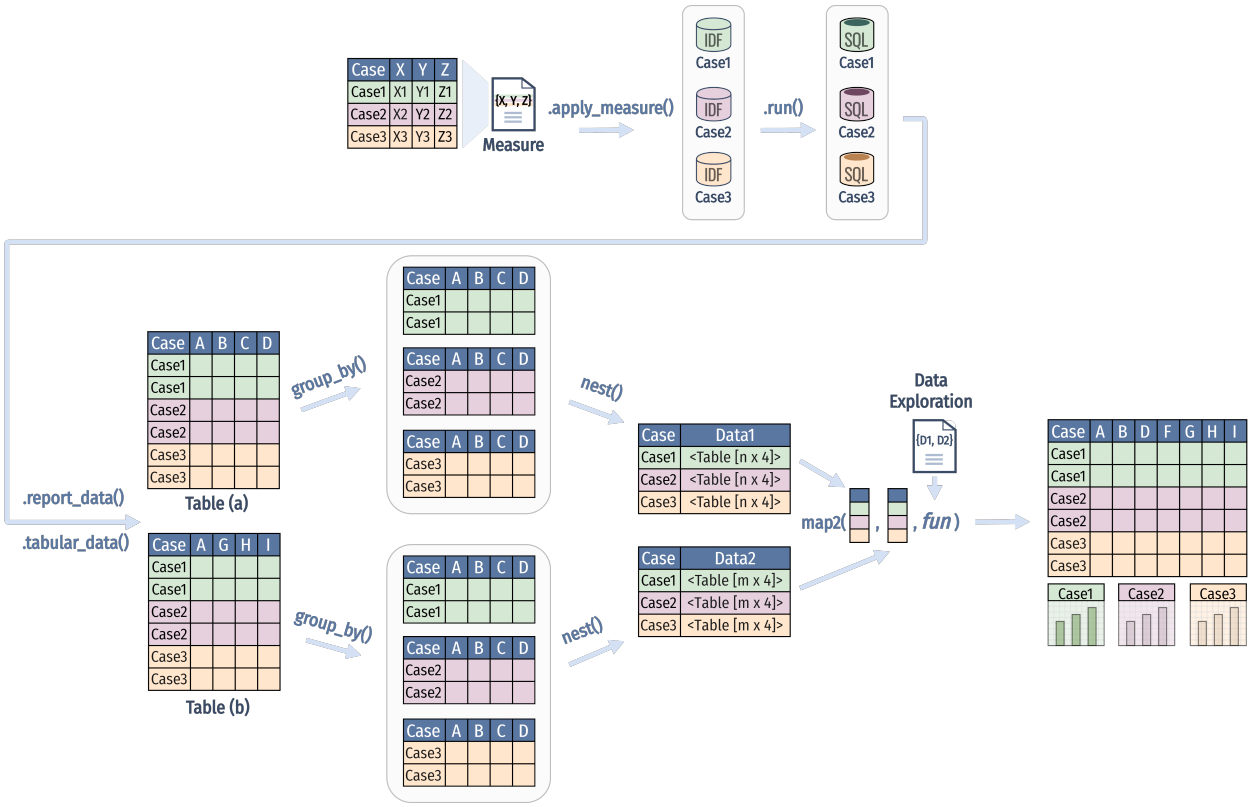

Figure 11: Workflow overview of parametric simulation analysis using the proposed parametric prototype

data structure makes it effortless to perform comparative analyses by taking the code snippets developed in data exploration for a single simulation and applying them to each of the parametric simulations. In this example, most of the EUI breakdown calculation code in Listing 1 have been reused (lines 70 – 77 in 2). It also demonstrates how to use the `case` column to perform case filtering (line 80 in Listing 2), table joins, and grouped summarization (lines 84 – 85 in Listing 2).

Fig. 12 shows the energy savings of various lighting technologies and plug loads control strategies, based on lines 82 – 96 in Listing 2. All technologies show overall energy savings to various degrees. Using higher efficiency lightings shows promising savings in both reducing the lighting electricity usage, with T5 and LED saving 34.9%% and 53.5% respectively, and the corresponding overall energy savings for T5 and LED are 7.5% and 11.4%. Strategies of turning off 40% and an 80% unnecessary plug loads during off-work hours reduce 11.3% and 22.6% electricity usage from interior equipment and improve the overall energy performance by 3.0% and 5.8%, respectively. Additional energy savings can be obtained when incorporating LED with an 80% reduction factor in off-work plug loads. However, even the overall energy savings are positive for all cases, trend for heating energy shows the opposite. This is due to the reason that all examined technologies will reduce indoor heat gains which plays a positive role during heating seasons.

4.3. Multi-objective optimization using Genetic Algorithm

Automated optimization has become increasingly popular in BES research and applications to efficiently search and identify optimal or near-optimal design options meeting one or more key design performance objectives [8]. Multi-objective optimization has also shown its potentials in building energy simulation calibration [63]. However, existing BES frameworks and applications for MOO often have constraints in the number of optimization objectives and limited flexibility in optimization parameter specifications.

The epluspar R package is an extension of the eplusr R package. It implements a `GAOptimJob` class which

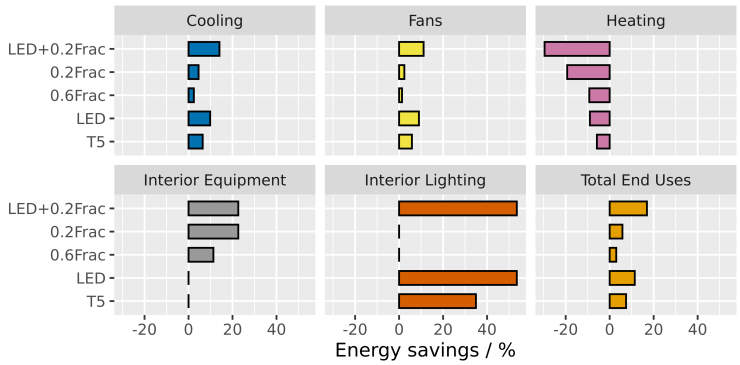

Figure 12: Energy savings of various lighting technologies and plug loads control strategies

is based on the parametric prototype and the *ecr* R package [64] for solving BES optimization problems using the Genetic Algorithm (GA). The `GAOptimJob` class leverages the proposed framework in terms of data structure and parametric simulation management and is capable of defining any number of arbitrary customized objective functions. It implements flexible general-purpose GA interfaces to solve BES-based single- or multi-objective optimization problems. This example demonstrates how to use the `GAOptimJob` class to solve a MOO problem, i.e. reducing carbon emissions and discomfort hours of the medium office reference building at the same time, by varying (1) indoor heating and cooling setpoint temperatures, (2) window-to-wall ratio (WWR) and (3) exterior wall insulation thickness. Fig. 13 gives an overview of a typical GA-based MOO process using the `GAOptimJob` class. Listing 3 shows the actual R code. The process shown in Fig. 13 can be divided into four main parts:

1. Specify optimization parameters
2. Create optimization objective functions
3. Set GA operators
4. Gather results and perform further analyses

Built on top of the parametric prototype, `GAOptimJob` class provides a similar interface for parametric model generation in the `apply_measure()` method. It can take the same measure functions to describe how optimization parameters should be modified. Lines 11 – 62 in Listing 3 define three measure functions to accept various design options in terms of (1) indoor heating and cooling set point, (2) window-to-wall ratio (WWR) and (3) exterior wall insulation thickness. The actual optimization parameter values in each generation are automatically calculated based on the GA operators and provided to `apply_measure()` method for parametric model generation.

`GAOptimJob` provides the flexibility to define objective functions of an optimization problem using any results from the simulation outputs. The `objective()` method takes objective definitions, evaluates them after simulations, and extracts the fitness together with optimization parameter values into a tidy table for post-processing using GA operators. In this example, Lines 88 – 95 in and lines 97–104 in Listing 3 define functions to extract the annual total carbon emissions and discomfort hours counted based on the Standard ASHRAE 55 – 2004 from the standard reports using tidy data interface. The `objective()` method in Line 107 in Listing 3 takes these two objective functions and tells the algorithm the minimization optimization direction.

`GAOptimJob` class has three key genetic operators (methods): (1) `selector()` (to select individuals to breed a new generation), (2) `mutator()` (to alter parts of one solution randomly), and (3) `recombinator()` (also called crossover, to swap parts of the solution with another), providing detailed procedures and steps on how to generate children from parent solutions. Lines 114 – 118 in Listing 3 directly specify those three operators with the default values that are tweaked to directly perform MOO using the Non-Dominated Sorting Genetic Algorithm (NSGA-II). The `terminator()` method is used to specify conditions to terminate the computation. In this example, we set it to stop when one hundred generations have been evaluated.

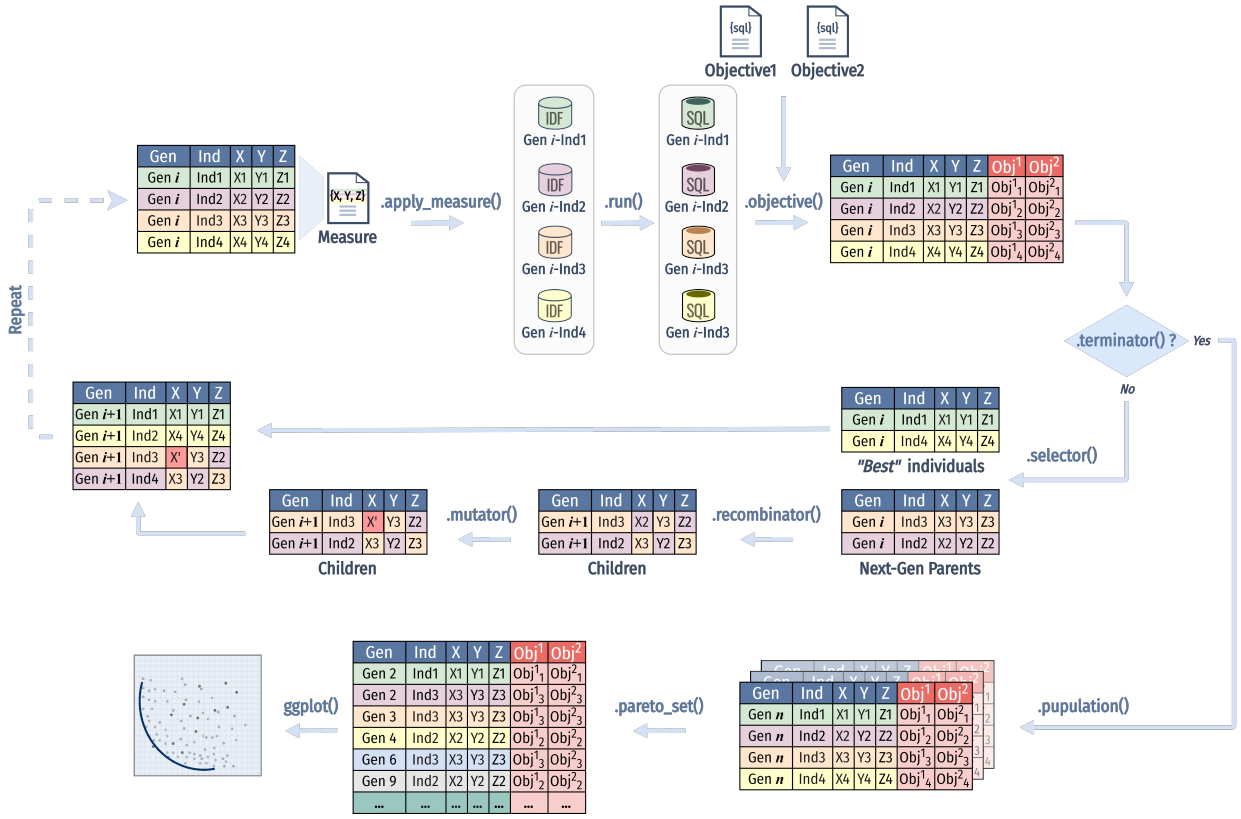

Figure 13: Workflow overview of performing multi-objective optimization on an EnergyPlus model using the epluspar package that is based on the proposed parametric prototype

With all objectives, parameters, and operators specified, the optimization will start with the `run()` method. In this example, we have twenty individuals per generation, resulting in a total of two thousand annual energy simulations. Once one of the conditions specified in `terminator()` is met, all populations and Pareto set can be extracted into two tidy tables for further analyses, using the `population()` and `pareto_set()` method (Lines 126 and 129 in Listing 3). Fig. 14 shows the Pareto front of discomfort hours and total carbon emissions generated using lines 132 – 139 in Listing 3. The final Pareto font contained 20 unique solutions.

Fig. 15 shows the parallel coordinates charts of the Pareto set. The carbon emissions have seen a significant reduction from the original value of 290ton. However, there were 10 out of 20 solutions in the Pareto set that performed worse in terms of providing a satisfactory indoor thermal environment. One possible solution to avoid this is to add a constraint when evaluating the fitness of the `discomfort_hours` objective, making sure all solutions that have larger discomfort hours should be abandoned.

4.4. Bayesian calibration

Model calibration is an essential process to achieve greater confidence in BES results. In recent years there has been an increasing application of Bayesian approaches for BES calibration [1]. Bayesian calibration is carried out following the statistical formulation proposed by Kennedy and O'Hagan [65]. This example demonstrates the model calibration workflow using the epluspar R package. The epluspar R package implements the Bayesian calibration algorithm proposed by Chong [1] and guidelines proposed by Chong and Menberg [65], and encapsulates them into the `BayesCalibJob` class. The `BayesCalibJob` class inherits from the parametric prototype and thus can leverage all the parametric simulation management capabilities.

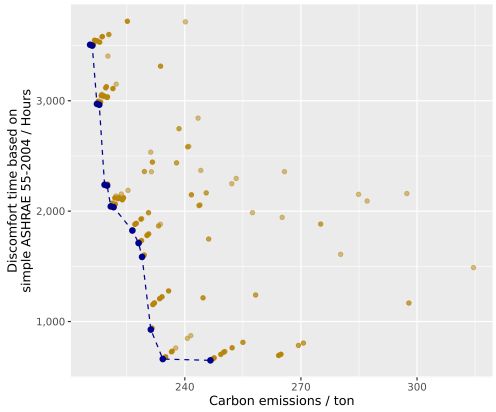

Figure 14: Pareto front of discomfort hours and carbon emissions

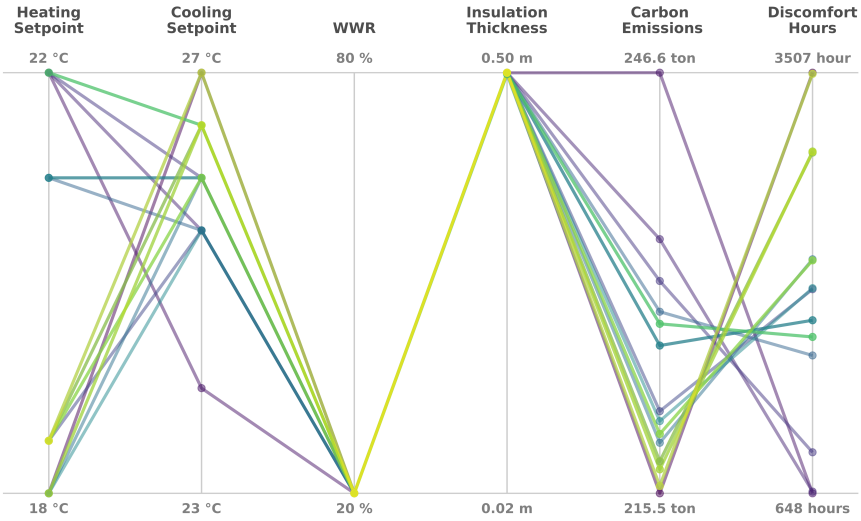

Figure 15: Parallel coordinates chart of the Pareto set

Fig. 16 gives an overview of a typical workflow of Bayesian calibration using the `BayesCalibJob` class in the epluspar package. Specifically, Listing 4 shows the workflow of calibrating one VAV fan total efficiency in the medium office reference model using observed fan air flow rate and electrical power.

The initial step for a Bayesian calibration is to collect data for observable input and output. In this example, since the reference model represents a virtual building with no measured data, we created some synthetic data for the examined period of July 1st to July 3rd using simulations (lines 1 – 27 in Listing 4). Also, the TMY3 weather data was used, instead of the Actual Meteorological Year (AMY) weather data. In real practice, the actual measurable variables may not be directly representable in EnergyPlus. In this case, an essential mapping process has to be performed to transform measured variables into EnergyPlus output variables (listed in RDD) and meters (listed in MDD), and connect the transformed measured values with the model using schedule files or other techniques. The next step is to specify the observable input and output variables for the calibration using the `input()` and `output()` methods in `BayesCalibJob` class (lines 29 – 39 in Listing 4).

Following the Bayesian calibration guidelines described in [65], the epluspar R package also introduces a `SensitivityJob` class based on the parametric prototype to perform calibration parameter screening with the Morris method. In `BayesCalibJob` class, the calibration parameters, together with the number of

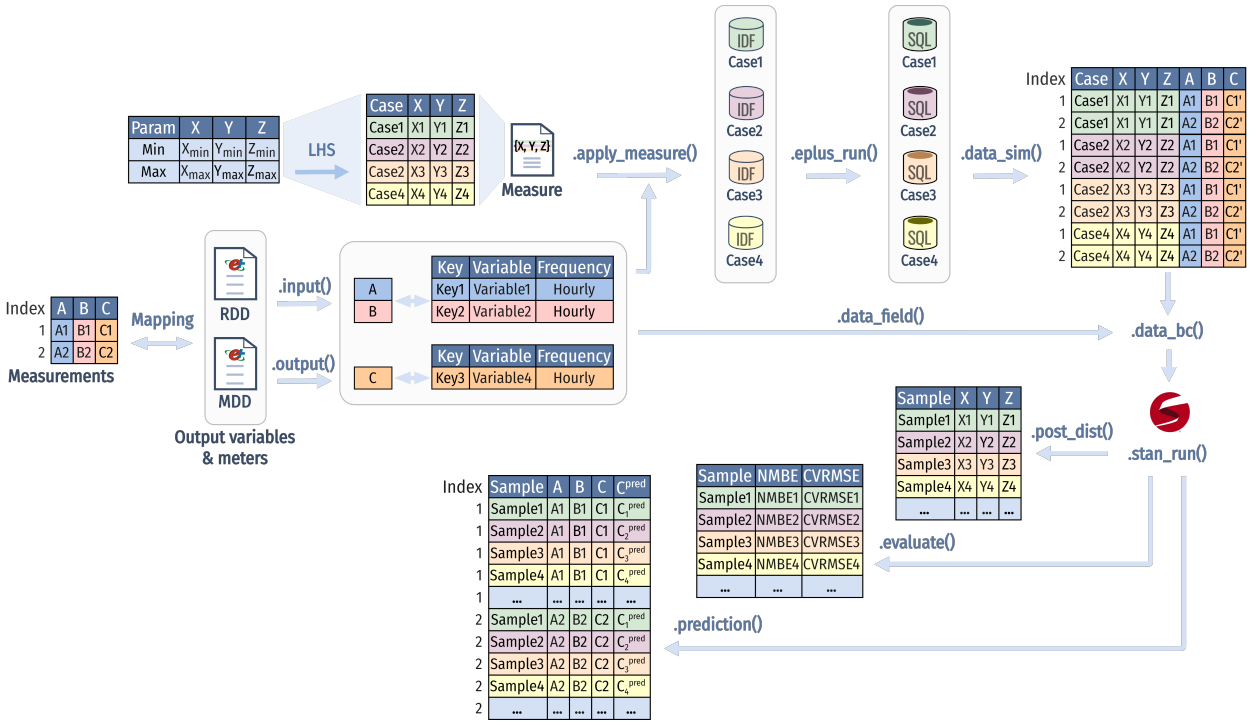

Figure 16: Workflow overview of performing Bayesian calibration on an EnergyPlus model using the epluspar package that is based on the proposed parametric prototype

EnergyPlus simulations, can be described using either the `param()` method (lines 41 – 45 in Listing 4) or the `apply_measure()` method. Each calibration parameter should be given a lower and upper bound value. Once the calibration parameters are given, the Latin Hypercube Sampling (LHS) algorithm will be used to generate parameter sample values based on the lower and upper bound and the simulation number (lines 47 – 48 in Listing 4). The benefit of LHS is that it will try to cover as much as possible in the multi-dimensional space of the calibration parameters.

After completing all simulations using the `eplus_run()` method (lines 50 – 51 in Listing 4), the `data_sim()` method gathers all simulated data of calibration input and output variables and aggregates the data into the same time frequency as the actual measured data (lines 53 – 54 in Listing 4). Method `data_bc()` will combine measured data and simulated data, and transform them into a list with the proper format for the Bayesian calibration algorithm (lines 56 – 60 in Listing 4) written in probabilistic programming language Stan. With all data required specified, the `stan_run()` method is used to call the Stan program (lines 62 – 63 in Listing 4). Once completed, the posterior distributions of calibration parameters and predicted output variable values can be retrieved using method `post_dist()` and `prediction()`, respectively (lines 66 – 70 in Listing 4). The uncertainty statistical indicators, including Normalized Mean Biased Error (NMBE) and Coefficient of Variation of the Root Mean Squared Error (CVRMSE) can be retrieved using the `evaluate()` method. Each method returns a tidy table that is well fit for the tidyverse data pipeline for data transformation and visualization.

Fig. 17 gives a density plot showing the posterior distributions of calibrated fan total efficiency, created using lines 74 – 79 in Listing 4. The mean value of the posterior distribution was 0.60, which is quite close to the actual value of 0.59 specified in the original model. Fig. 18 gives a box plot showing the distribution of CVRMSE and NMBE per Markov Chain Monte Carlo (MCMC) sampling, created using lines 81 – 85 in Listing 4. The mean NMBE value was quite close to zero, and the average CVRMSE is around 3.0%. Both values met the thresholds of CVRMSE $\leq$ 15% and NMBE $\leq$ 5% set by ASHRAE [66]. The satisfactory results are expected since we use synthetic data. However, the overall workflow shown in this example can

be applied to Bayesian calibration on building energy models with real measured data.

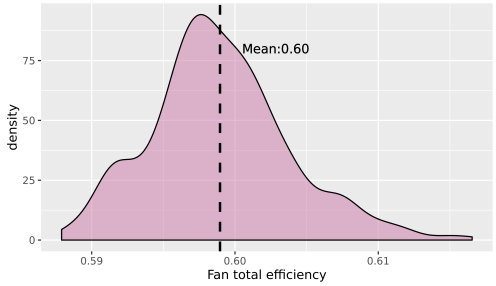

Figure 17: Posterior distribution of fan total efficiency

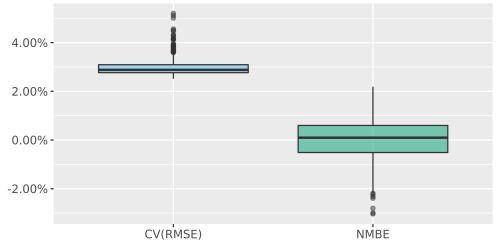

Figure 18: Distribution of CVRMSE and NMBE per MCMC sample

5. Limitations

The main limitation of the proposed framework lies in its R-oriented workflows, which currently may not be widely adopted in industry fields. Prospective users of the framework who do not know R must spend time learning how to use it. This drawback may be compensated by the growing user community.

Since the eplusr framework is mainly focused on modifying existing BES models, instead of creating new ones from scratch, currently it has limited capacities to perform sophisticated geometry transformation, including surface matching and rotation. Operations such as creating or replacing one whole HVAC system may also be time-consuming processes.

6. Conclusion

Building energy simulation (BES) has been widely adopted for the investigation of environmental and energy performance for different design and retrofit alternatives. The absence of seamless integration of BES and data-centric analysis raises problems in both the productivity and also the credibility of BES studies. This paper proposed a holistic framework called 'eplusr' to bridge the gap between the building energy simulation and data science domains.

Eplusr differs from existing frameworks by its data-centric design philosophy. It provides a tidy data interface for BES that matches the semantics of the simulation results with the data representation. The tidy data interface provides the possibility to query BES outputs with various types of specifications, which makes it easy and straightforward to extract simulation results from any time period and for any variables in a consistent manner. The tidy-formatted results can be easily fed to various data-centric analytics using existing tools in R.

The parametric prototype developed in this framework provides a set of abstractions to ease the process of parametric model generation, design alternative evaluation, and large parametric simulation management.

It is capable of defining various analyses using any algorithms available in R. The flexibility and extensibility of the parametric simulation prototype in this framework are demonstrated by its easy adoption to perform multi-objective optimization and Bayesian calibration.

The need for reproducibility in BEM is growing significantly, together with the ongoing trend of the increasing complexity of BEM projects. The eplusr framework provides a solution by developing a portable and reusable BES computational environment based on Docker containerization, encapsulating the toolchains for statistical computing, building energy modeling, data analytics, and literate programming. The open-source nature of the proposed framework will advocate the BES domain to embrace the tools essential for maintaining a reproducible workflow.

Supplementary materials

The supplementary files include code and datasets used in this article. More is available at https://github.com/ideas-lab-nus/eplusr-paper.

Acknowledgements

This research was funded by the Republic of Singapore's National Research Foundation through a grant to the Berkeley Education Alliance for Research in Singapore (BEARS) for the Singapore-Berkeley Building Efficiency and Sustainability in the Tropics (SinBerBEST) Program. BEARS has been established by the University of California, Berkeley as a center for intellectual excellence in research and education in Singapore.

References

[1] A. Chong, K.P. Lam, M. Pozzi, J. Yang, Bayesian calibration of building energy models with large datasets, Energy and Buildings. 154 (2017) 343–355. https://doi.org/10.1016/j.enbuild.2017.08.069.

[2] J. Kneifel, Life-cycle carbon and cost analysis of energy efficiency measures in new commercial buildings, Energy and Buildings. 42 (2010) 333–340. https://doi.org/10.1016/j.enbuild.2009.09.011.

[3] T. Hong, J. Langevin, K. Sun, Building simulation: Ten challenges, Building Simulation. 11 (2018) 871–898. https://doi.org/10.1007/s12273-018-0444-x.

[4] T. Hong, S.K. Chou, T.Y. Bong, Building simulation: An overview of developments and information sources, Building and Environment. 35 (2000) 347–361. https://doi.org/10.1016/S0360-1323(99)00023-2.

[5] D.B. Crawley, J.W. Hand, M. Kummert, B.T. Griffith, Contrasting the capabilities of building energy performance simulation programs, Building and Environment. 43 (2008) 661–673. https://doi.org/10.1016/j.buildenv.2006.10.027.

[6] H. Kim, A. Stumpf, W. Kim, Analysis of an energy efficient building design through data mining approach, Automation in Construction. 20 (2011) 37–43. https://doi.org/10.1016/j.autcon.2010.07.006.

[7] C. Miller, C. Hersberger, M. Jones, Automation of Common Building Energy Simulation Workflows Using Python, in: Proceedings of BS2013, Chambéry, France, 2013.

[8] S. Attia, M. Hamdy, W. O'Brien, S. Carlucci, Assessing gaps and needs for integrating building performance optimization tools in net zero energy buildings design, Energy and Buildings. 60 (2013) 110–124. https://doi.org/10.1016/j.enbuild.2013.01.016.

[9] C. Srivastava, Z. Yang, R.K. Jain, Understanding the adoption and usage of data analytics and simulation among building energy management professionals: A nationwide survey, Building and Environment. 157 (2019) 139–164. https://doi.org/10.1016/j.buildenv.2019.04.016.

[10] K. Fleming, N. Long, A. Swindler, Building Component Library: An Online Repository to Facilitate Building Energy Model Creation; Preprint, National Renewable Energy Lab. (NREL), Golden, CO (United States), 2012. https://www.osti.gov/biblio/1045093 (accessed May 7, 2020).

[11] T. Østergård, R.L. Jensen, S.E. Maagaard, Building simulations supporting decision making in early design A review, Renewable and Sustainable Energy Reviews. 61 (2016) 187–201. https://doi.org/10.1016/j.rser.2016.03.045.

[12] D.B. Crawley, L.K. Lawrie, F.C. Winkelmann, W.F. Buhl, Y.J. Huang, C.O. Pedersen, R.K. Strand, R.J. Liesen, D.E. Fisher, M.J. Witte, J. Glazer, EnergyPlus: Creating a new-generation building energy simulation program, Energy and Buildings. 33 (2001) 319–331. https://doi.org/10.1016/s0378-7788(00)00114-6.

[13] W.A. Beckman, L. Broman, A. Fiksel, S.A. Klein, E. Lindberg, M. Schuler, J. Thornton, TRNSYS The most complete solar energy system modeling and simulation software, Renewable Energy. 5 (1994) 486–488. https://doi.org/10.1016/0960-1481(94)90420-0.

[14] D. Yan, J. Xia, W. Tang, F. Song, X. Zhang, Y. Jiang, DeST An integrated building simulation toolkit Part I: Fundamentals, Building Simulation. 1 (2008) 95–110. https://doi.org/10.1007/s12273-008-8118-8.

[15] J.W. Hand, The ESP-r cookbook, Energy Systems Research Unit, Department of Mechanical and Aerospace Engineering University of Strathclyde, Glasgow, UK, 2018.

[16] Integrated Environmental Solutions Limited, IES Virtual Environment, (2020). https://www.iesve.com/software/virtual-environment (accessed June 30, 2020).

[17] T. Kalamees, IDA ICE: The simulation tool for making the whole building energy- and HAM analysis., in: Working Meeting of Annex 41 MOIST-ENG, Zurich, Switzerland, 2004: p. 6.

[18] J.J. Hirsch, eQUEST: The QUick Energy Simulation Tool, (2020). http://www.doe2.com/equest/ (accessed July 1, 2020).

[19] DesignBuilder Software Ltd, DesignBuilder, (2020). https://designbuilder.co.uk/ (accessed July 1, 2020).

[20] R. Guglielmetti, D. Macumber, N. Long, OpenStudio: An Open Source Integrated Analysis Platform; Preprint, National Renewable Energy Lab. (NREL), Golden, CO (United States), 2011. https://www.osti.gov/biblio/1032670 (accessed March 8, 2020).

[21] Z. Yi, jEplus, (2020). http://www.jeplus.org/wiki/doku.php (accessed March 9, 2020).

[22] Big Ladder Software, Modelkit/Params Framework, (2020). https://bigladdersoftware.com/projects/modelkit/ (accessed March 9, 2020).

[23] M. Wetter, GenOpt – A Generic Optimization Program, in: Proceedings of IBPSA's Building Simulation 2001 Conference, Rio de Janeiro, 2001: p. 9.

[24] M.S. Roudsari, M. Pak, A. Smith, Ladybug: A parametric environmental plugin for grasshopper to help designers create an environmentally-conscious design, in: Proceedings of BS2013, 2013.

[25] A. Tabadkani, M. Valinejad Shoubi, F. Soflaei, S. Banihashemi, Integrated parametric design of adaptive facades for user's visual comfort, Automation in Construction. 106 (2019) 102857. https://doi.org/10.1016/j.autcon.2019.102857.

[26] S. Philip, Eppy: Scripting language for E+, Energyplus, (2020). http://www.github.com/santoshphilip/eppy/.

[27] W. Bernal, M. Behl, T.X. Nghiem, R. Mangharam, MLE+: A tool for integrated design and deployment of energy efficient building controls, in: Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings, Association for Computing Machinery, Toronto, Ontario, Canada, 2012: pp. 123–130. https://doi.org/10.1145/2422531.2422553.

[28] J. Zhao, K.P. Lam, B.E. Ydstie, EnergyPlus Model-Based Predictive Control (EPMPC) by Using Matlab/Simulink and MLE+, in: Proceedings of BS2013, Chambéry, France, 2013: pp. 2466–2473.

[29] P.G. Schild, EpXL: EnergyPlus-Excel, (2020). https://github.com/SchildCode/EpXL (accessed June 2, 2020).

[30] P.B. Purup, S. Petersen, Research framework for development of building performance simulation tools for early design stages, Automation in Construction. 109 (2020) 102966. https://doi.org/10.1016/j.autcon.2019.102966.

[31] D. Macumber, Scripted Building Energy Modeling and Analysis, National Renewable Energy Lab. (NREL), Golden, CO (United States), 2012. https://www.osti.gov/biblio/1053759 (accessed May 7, 2020).

[32] A. Roth, J. Bull, S. Criswell, P. Ellis, J. Glazer, D. Goldwasser, N. Kruis, A. Parker, S. Philip, D. Reddy, Scripting frameworks for enhancing energyplus modeling productivity, in: Proceedings of SimBuild 2018, 2018: p. 8.

[33] A. Parker, K. Benne, L. Brackney, E. Hale, D. Macumber, M. Schott, E. Weaver, A Parametric Analysis Tool for Building Energy Design Workflows: Application to a Utility Design Assistance Incentive

Program, ACEEE Summer Study on Energy Efficiency in Buildings. (2014) 12.

[34] Y. Zhang, I. Korolija, Performing complex parametric simulations with jEPlus, in: Proceedings of SET2010, Shanghai, China, 2010: p. 5.

[35] B. Kiss, Z. Szalay, Modular approach to multi-objective environmental optimization of buildings, Automation in Construction. 111 (2020) 103044. https://doi.org/10.1016/j.autcon.2019.103044.

[36] M. Molina-Solana, M. Ros, M.D. Ruiz, J. Gómez-Romero, M.J. Martin-Bautista, Data science for building energy management: A review, Renewable and Sustainable Energy Reviews. 70 (2017) 598–609. https://doi.org/10/f928fw.

[37] H. Burak Gunay, W. Shen, G. Newsham, Data analytics to improve building performance: A critical review, Automation in Construction. 97 (2019) 96–109. https://doi.org/10.1016/j.autcon.2018.10.020.

[38] R Core Team, R: A language and environment for statistical computing, (2019). https://www.R-project.org/.

[39] T.E. Oliphant, Python for Scientific Computing, Computing in Science Engineering. 9 (2007) 10–20. https://doi.org/10.1109/MCSE.2007.58.

[40] J.S.S. Lowndes, B.D. Best, C. Scarborough, J.C. Afflerbach, M.R. Frazier, C.C. O'Hara, N. Jiang, B.S. Halpern, Our path to better science in less time using open data science tools, Nature Ecology & Evolution. 1 (2017) 1–7. https://doi.org/10.1038/s41559-017-0160.

[41] R. Peng, The reproducibility crisis in science: A statistical counterattack, Significance. 12 (2015) 30–32. https://doi.org/10.1111/j.1740-9713.2015.00827.x.

[42] C. Boettiger, An introduction to Docker for reproducible research, ACM SIGOPS Operating Systems Review. 49 (2015). https://doi.org/10.1145/2723872.2723882.

[43] M. Baker, 1,500 scientists lift the lid on reproducibility, Nature News. 533 (2016) 452. https://doi.org/10.1038/533452a.

[44] D. Merkel, Docker: Lightweight Linux containers for consistent development and deployment, Linux Journal. 2014 (2014) 2:2. https://doi.org/10.5555/2600239.2600241.

[45] E.F. Codd, The relational model for database management: Version 2, Addison-Wesley Longman Publishing Co., Inc., USA, 1990.

[46] Wikipedia, Object-oriented programming, Wikipedia. (2020). https://en.wikipedia.org/w/index.php?title=Object-oriented_programming (accessed March 7, 2020).

[47] M. Dowle, A. Srinivasan, J. Gorecki, M. Chirico, P. Stetsenko, T. Short, S. Lianoglou, E. Antonyan, M. Bonsch, H. Parsonage, S. Ritchie, K. Ren, X. Tan, R. Saporta, O. Seiskari, X. Dong, M. Lang, W. Iwasaki, S. Wenchel, K. Broman, T. Schmidt, D. Arenburg, E. Smith, F. Cocquemas, M. Gomez, P. Chataignon, D. Groves, D. Possenriede, F. Parages, D. Toth, M. Yaramaz-David, A. Perumal, J. Sams, M. Morgan, M. Quinn, @javrucebo, @marc-outins, R. Storey, M. Saraswat, M. Jacob, M. Schubmehl, D. Vaughan, Data.Table: Extension of 'data.frame', (2019). https://CRAN.R-project.org/package=data.table (accessed March 8, 2020).

[48] H. Wickham, Tidy Data, Journal of Statistical Software. 59 (2014) 1–23. https://doi.org/10.18637/jss.v059.i10.

[49] H. Wickham, G. Grolemund, R for Data Science: Import, Tidy, Transform, Visualize, and Model Data, 1 edition, O'Reilly Media, Sebastopol, CA, 2017.

[50] H. Wickham, M. Averick, J. Bryan, W. Chang, L. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. Pedersen, E. Miller, S. Bache, K. Müller, J. Ooms, D. Robinson, D. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, H. Yutani, Welcome to the Tidyverse, Journal of Open Source Software. 4 (2019) 1686. https://doi.org/10.21105/joss.01686.

[51] M. Owens, The definitive guide to SQLite, Apress, Berkeley, Calif, 2006.

[52] M.D. Morris, Factorial Sampling Plans for Preliminary Computational Experiments, Technometrics. 33 (1991) 161–174. https://doi.org/10.1080/00401706.1991.10484804.

[53] D. Nüst, D. Eddelbuettel, D. Bennett, R. Cannoodt, D. Clark, G. Daroczi, M. Edmondson, C. Fay, E. Hughes, L. Kjeldgaard, S. Lopp, B. Marwick, H. Nolis, J. Nolis, H. Ooi, K. Ram, N. Ross, L. Shepherd, P. Sólymos, T.L. Swetnam, N. Turaga, J. Williams, C. Willis, N. Xiao, C. Van Petegem, The Rockerverse: Packages and Applications for Containerization with R, arXiv:2001.10641 [Cs]. (2020). http://arxiv.org/abs/2001.10641 (accessed March 8, 2020).

[54] C. Boettiger, D. Eddelbuettel, An Introduction to Rocker: Docker Containers for R, The R Journal. 9 (2017) 527–536. https://doi.org/10.32614/RJ-2017-065.

[55] D.E. Knuth, Literate Programming, The Computer Journal. 27 (1984) 97–111. https://doi.org/10.1093/comjnl/27.2.97.

[56] Y. Xie, J.J. Allaire, G. Grolemund, R Markdown: The Definitive Guide, Chapman and Hall/CRC, Boca Raton, Florida, 2018. https://bookdown.org/yihui/rmarkdown/ (accessed May 28, 2020).

[57] T. Kluyver, B. Ragan-Kelley, F. Pérez, M. Bussonnier, J. Frederic, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, S. Abdalla, C. Willing, Jupyter Notebooks: A publishing format for reproducible computational workflows, in: Positioning and Power in Academic Publishing: Players, Agents and Agendas, 2016: pp. 87–90. https://doi.org/10.3233/978-1-61499-649-1-87.

[58] Y. Xie, Dynamic Documents with R and knitr, Second Edition, 2 edition, Routledge, Boca Raton, 2015.

[59] A. Krewinkel, R. Winkler, Formatting Open Science: Agilely creating multiple document formats for academic manuscripts with Pandoc Scholar, PeerJ Computer Science. 3 (2017) e112. https://doi.org/10.7717/peerj-cs.112.

[60] J.J. Allaire, Y. Xie, J. McPherson, J. Luraschi, K. Ushey, A. Atkins, H. Wickham, J. Cheng, W. Chang, R. Iannone, A. Dunning, A. Yasumoto, B. Schloerke, C. Dervieux, F. Aust, J. Allen, J. Seo, M. Barrett, R. Hyndman, R. Lesur, R. Storey, R. Arslan, S. Oller, RStudio Inc, Rmarkdown: Dynamic Documents for R, (2020). https://CRAN.R-project.org/package=rmarkdown (accessed May 28, 2020).

[61] Y. Xie, TinyTeX: A lightweight, cross-platform, and easy-to-maintain LaTeX distribution based on TeX Live, TUGboat. (2019) 30–32. http://tug.org/TUGboat/Contents/contents40-1.html.

[62] K. Field, M. Deru, D. Studer, Using DOE Commercial Reference Buildings for Simulation Studies: Preprint, National Renewable Energy Lab. (NREL), Golden, CO (United States), 2010. https://www.osti.gov/biblio/988604 (accessed May 29, 2020).

[63] Z. Yang, B. Becerik-Gerber, A model calibration framework for simultaneous multi-level building energy simulation, Applied Energy. 149 (2015) 415–431. https://doi.org/10/f7fnf7.

[64] J. Bossek, Ecr 2.0: A modular framework for evolutionary computation in R, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, ACM, Berlin Germany, 2017: pp. 1187–1193. https://doi.org/10.1145/3067695.3082470.

[65] A. Chong, K. Menberg, Guidelines for the Bayesian calibration of building energy models, Energy and Buildings. 174 (2018) 527–547. https://doi.org/10/gd5s5b.

[66] ASHRAE, Guideline 14-2014, Measurement of Energy and Demand Savings, (2014).

Appendix A. Code for data exploration

```r
# install packages
install.packages(c("eplusr", "tidyverse"))

# load package
library(eplusr)
library(tidyverse) # for data-driven analytics

# get EnergyPlus v9.1 installation directory
dir <- eplus_config(9.1)$dir

# use example model and weather file distributed with EnergyPlus v9.1
path_model <- file.path(dir, "ExampleFiles/RefBldgMediumOfficeNew2004_Chicago.idf")
path_weather <- file.path(dir, "WeatherData/USA_IL_Chicago-OHare.Intl.AP.725300_TMY3.epw")

# read model
idf <- read_idf(path_model)

##############
# Model API #
##############

# make sure weather file input is respected
idf$SimulationControl$Run_Simulation_for_Weather_File_Run_Periods <- "Yes"

# make sure energy consumption is presented in kWh
idf$OutputControl_Table_Style$Unit_Conversion <- "JtoKWH"

# save the modified model into a temporary folder
idf$save(file.path(tempdir(), "MediumOffice.idf"))

# run annual simulation
job <- idf$run(path_weather)

########################
# Tidy data interface #
########################

# read building area from Standard Reports
area <- job$tabular_data(table_name = "Building Area", wide = TRUE)[[1L]]

# read building energy consumption from Standard Reports
end_use <- job$tabular_data(table_name = "End Uses", wide = TRUE)[[1L]]

# read zone metadata from Standard Input and Output
zones <- job$read_table("Zones")

# read hourly air-conditioning system output with all additional metadata for
# the annual simulation from Variable Output
aircon_out <- job$report_data(
    name = sprintf("air system total %s energy", c("heating", "cooling")),
    environment_name = "annual",
    all = TRUE
)

```

```r
##########################
# Data-driven analytics #
##########################

# calculate Energy Use Intensity (EUI) for electricity
eui <- end_use %>%
    # only select columns of interest
    select(category = row_name, electricity = `Electricity [kWh]`) %>%
    # get rid of category with empty energy consumption
    filter(electricity > 0.0) %>%
    # order by value
    arrange(-electricity) %>%
    # calculate EUI
    mutate(eui = round(electricity / area$'Area [m2]'[1], digits = 2)) %>%
    # calculate proportion of each category
    mutate(proportion = round(eui / eui[1] * 100, digits = 2)) %>%
    # remove electricity column
    select(-electricity)

# plot a pie chart to show EUI breakdown
p_eui <- eui %>%
    filter(category != "Total End Uses") %>%
    mutate(category = as_factor(sprintf("%s [%.2f%%]", category, proportion, "%"))) %>%
    ggplot(aes("", proportion, fill = category)) +
    geom_bar(stat = "identity", width = 1, color = "black", size = 0.2) +
    coord_polar("y", start = 0)

# calculate air-conditioned floor area per storey
storey <- zones %>%
    # exclude plenum zones
    filter(is_part_of_total_area == 1) %>%
    # group by centroid height
    group_by(centroid_height = round(centroid_z, digits = 4)) %>%
    # calculate total floor area
    summarise(floor_area = sum(floor_area)) %>%
    ungroup() %>%
    # add storey index
    arrange(centroid_height) %>%
    mutate(storey = seq_len(n()), air_system = paste("VAV", storey, sep = "_")) %>%
    select(air_system, floor_area)

# get monthly heating and cooling demands per served area
aircon_out_mon <- aircon_out %>%
    # only consider weekdays
    filter(!day_type %in% c("Holiday", "Saturday", "Sunday")) %>%
    # add an identifier column to indicate cooling and heating condition
    mutate(type = case_when(
        str_detect(name, "Heating") ~ "Heating",
        str_detect(name, "Cooling") ~ "Cooling"
    )) %>%
    # add floor area served by each air-conditioning system
    left_join(storey, c("key_value" = "air_system")) %>%
    # calculate the monthly averaged heating and cooling demands in MJ/m2
    group_by(month, type, air_system = key_value) %>%
    summarise(system_output = sum(value) / 1e6 / floor_area[1]) %>%
    ungroup()
```

```r

# plot a pie chart to show the heating and cooling demand profile
p_aircon_out <- aircon_out_mon %>%
    mutate(month = as.factor(month)) %>%
    mutate(system_output = case_when(
        type == "Heating" ~ -system_output,
        type == "Cooling" ~ system_output
    )) %>%
    ggplot() +
    geom_col(aes(month, system_output, group = type, fill = type), position = "dodge") +
    facet_wrap(vars(air_system), ncol = 1) +
    labs(x = "", y = "Heating and cooling demands / MJ m-2")
```

Listing 1: Data exploration on the EUI and the heating and cooling demands

Appendix B. Code for parametric simulation

```r
# create a parametric prototype of given model and weather file
param <- param_job(idf, path_weather)

######################
#  Create Measures  #
######################

# create a measure for modifying LPD
set_lpd <- function (idf, lpd = NA) {
    # keep the original if applicable
    if (is.na(lpd)) return(idf)

    # set 'Watts per Zone Floor Area' in all 'Lights' objects as input LPD
    idf$set(Lights := list(watts_per_zone_floor_area = lpd))

    # return the modified model
    idf
}

# create a measure for reducing plug loads during off-work time
set_nightplug <- function (idf, frac = NA) {
    # keep the original if applicable
    if (is.na(frac)) return(idf)

    # extract the plug load schedule into a tidy table
    sch <- idf$to_table("bldg_equip_sch")

    # modify certain schedule value specified using field names
    sch <- sch %>%
        mutate(value = case_when(
            field %in% paste("Field", c(4,14,16,18)) ~ sprintf("%.2f", as.numeric(value) * frac),
            TRUE ~ value
        ))

    # update schedule object using the tidy table
    idf$update(sch)

```

```r
    # return the modified model
    idf
}

# combine two measures into one
ecm <- function (idf, lpd, nightplug_frac) {
    idf %>% set_lpd(lpd) %>% set_nightplug(nightplug_frac)
}

#####################
#  Apply Measures  #
#####################

# apply measures and create parametric models
param$apply_measure(ecm,
                    lpd = c(   NA,  7.0,   5.0,        NA,        NA,           5.0),
       nightplug_frac = c(   NA,   NA,    NA,       0.6,       0.2,           0.2),
               # name of each case
               .names = c("Ori", "T5", "LED", "0.6Frac", "0.2Frac", "LED+0.2Frac")
)

# run parametric simulations in parallel
param$run()

#########################
# Data-driven analytics #
#########################

# read building energy consumption from Standard Reports
param_end_use <- param$tabular_data(table_name = "End Uses", wide = TRUE)[[1L]]

# calculate EUI breakdown
param_eui <- param_end_use %>%
    select(case, category = row_name, electricity = `Electricity [kWh]`) %>%
    filter(electricity > 0.0) %>%
    arrange(-electricity) %>%
    mutate(eui = round(electricity / area$'Area [m2]'[1], digits = 2)) %>%
    select(case, category, eui) %>%
    # exclude categories that did not change
    filter(category != "Pumps", category != "Exterior Lighting")

# extract the seed model, i.e. "Ori" case as the baseline
ori_eui <- param_eui %>% filter(case == "Ori") %>% select(-case)

# calculate energy savings based on the baseline EUI
param_savings <- param_eui %>%
    right_join(ori_eui, by = "category", suffix = c("", "_ori")) %>%
    mutate(savings = (eui_ori - eui) / eui_ori * 100) %>%
    filter(case != "Ori")

# plot a bar chart to show the energy savings
p_param_savings <- param_savings %>%
    mutate(case = factor(case, names(param$models()))) %>%
    ggplot(aes(case, savings, fill = category)) +
    geom_bar(position = "dodge", stat = "identity", width = 0.6, color = "black",
        show.legend = FALSE) +
```

```
    facet_wrap(vars(category), nrow = 2) +
    labs(x = NULL, y = "Energy savings / %") +
    coord_flip()
```

Listing 2: Parametric simulation of ECMs on plug loads and LPD

Appendix C. Code for multi-objective optimization using Genetic Algorithm

```
# load package
library(epluspar)

# create a GA optimization job
ga <- gaoptim_job(idf, path_weather)

############################
# Optimization variables  #
############################

# define a measure to change heating setpoint
set_heating_setpoint <- function (idf, sp) {
    sp <- as.character(sp)
    idf$set(htgsetp_sch = list(field_6 = sp, field_16 = sp, field_21 = sp))
    idf
}

# define a measure to change cooling setpoint
set_cooling_setpoint <- function (idf, sp) {
    sp <- as.character(sp)
    idf$set(clgsetp_sch = list(field_6 = sp, field_13 = sp))
    idf
}

# define a measure to change the window-to-wall ratio
set_wwr <- function (idf, wwr) {
    # extract data of all windows
    win <- idf$to_table(class = "FenestrationSurface:Detailed", wide = TRUE, string_value = FALSE)

    # extract data of all parent walls
    wall <- idf$to_table(win[["Building Surface Name"]], wide = TRUE,
        string_value = FALSE, group_ext = "index"
    )

    # calculate new X and Y coordinates for windows
    cols <- sprintf("Vertex %s-coordinate", c("X", "Y", "Z"))
    ratio <- c(0.999, 0.999, wwr)
    cal_coords <- function (coords, ratio) {
        list(round((coords[[1]] - mean(coords[[1L]])) * ratio + mean(coords[[1L]]), 3))
    }
    wall[, .SDcols = cols, by = "id", c(cols) := mapply(
        cal_coords, coords = .SD, ratio = ratio, SIMPLIFY = FALSE
    )]

    # update coordinates of windows
    coords <- wall[, lapply(.SD, unlist), .SDcols = cols, by = "id"]
```

```
    coords <- lapply(coords[, -"id"], function (x) as.data.frame(t(matrix(x, nrow = 4))))
    for (axis in c("X", "Y", "Z")) {
        cols <- sprintf("Vertex %i %s-coordinate", 1:4, axis)
        win[, c(cols) := coords[[sprintf("Vertex %s-coordinate", axis)]]]
    }

    idf$update(dt_to_load(win))

    idf
}

# define a measure to change the insulation thickness of the exterior wall
set_insulation <- function (idf, thickness) {
    idf$set(`Steel Frame NonRes Wall Insulation` = list(thickness = thickness))
    idf
}

# combine all measures into one
design_options <- function (idf, htg_sp, clg_sp, wwr, insulation_thickness) {
    idf <- set_heating_setpoint(idf, htg_sp)
    idf <- set_cooling_setpoint(idf, clg_sp)
    idf <- set_wwr(idf, wwr)
    idf <- set_insulation(idf, insulation_thickness)
    idf
}

# specify design space of parameters
ga$apply_measure(design_options,
    htg_sp = choice_space(seq(18, 22, 0.5)),
    clg_sp = choice_space(seq(23, 27, 0.5)),
    wwr = float_space(0.2, 0.8),
    insulation_thickness = float_space(0.02, 0.5)
)

# validate to make sure all measures and objective functions work properly
ga$validate(ddy_only = FALSE)

############################
# Optimization objectives  #
############################

# define an objective function to get carbon emissions
carbon_emissions <- function (idf) {
    as.double(idf$last_job()$tabular_data(
        report_name = "emissions data summary",
        row_name = "Annual Sum or average",
        column_name = "carbon equivalent:facility"
    )$value)
}

# define an objective function to get discomfort hours
discomfort_hours <- function (idf) {
    as.double(idf$last_job()$tabular_data(
        table_name = "comfort and setpoint not met summary",
        row_name = "time not comfortable based on simple ASHRAE 55-2004",
        column_name = "facility"
```

```
    )$value)
}

# set optimization objectives
ga$objective(carbon_emissions, discomfort_hours, .dir = "min")

#################
# GA operators  #
#################

# specify how to mix solutions
ga$recombinator()
# specify how to change parts of one solution randomly
ga$mutator()
# specify how to select best solutions
ga$selector()
# specify the conditions when to terminate the computation
ga$terminator(max_gen = 100L)

# run optimization
ga$run(mu = 20)

###############################################
# Gather results and perform further analyses  #
###############################################

# get all population
population <- ga$population()

# get Pareto set
pareto <- ga$pareto_set()

# plot Pareto front
p_pareto <- ggplot() +
    geom_point(aes(carbon_emissions, discomfort_hours), population, color = "darkgoldenrod", alpha = 0.5) +
    geom_line(aes(carbon_emissions, discomfort_hours), pareto, color = "darkblue", linetype = 2) +
    geom_point(aes(carbon_emissions, discomfort_hours), pareto, color = "darkblue", size = 2) +
    scale_x_continuous("Carbon emissions / ton", labels = scales::number_format(scale = 0.001)) +
    scale_y_continuous("Discomfort time based on\nsimple ASHRAE 55-2004 / Hours",
        labels = scales::number_format(big.mark = ",")
    )
```

Listing 3: Multi-objective optimization using Genetic Algorithm

Appendix D. Code for Bayesian calibration

```
#############################################################################
# NOTE: for demonstration, we use the seed model to generate some synthetic data
# clone the original model
tmp <- idf$clone()
# remove all existing run periods
tmp$RunPeriod <- NULL
# add a new run period from Jul 1st to Jul 3rd
tmp$add(RunPeriod = list("test", 7, 1, NULL, 7, 3))
```

```r
# add variables of interest to output
tmp$Output_Variable <- NULL
tmp$add(Output_Variable = list("VAV_1_Fan", "Fan Electric Power", "Hourly"))
# get rid of design day variable output data
tmp$SimulationControl$set(run_simulation_for_sizing_periods = "No")
# save the model to a temporary file
tmp$save(tempfile(fileext = ".idf"))
# run simulation
job <- tmp$run(path_weather)
# extract fan electric power in 6-hourly frequency
fan_power <- job$report_data(all = TRUE) %>%
    epluspar:::report_dt_aggregate("6 hour") %>%
    eplusr:::report_dt_to_wide()
# insert Gaussian noise
fan_power <- fan_power %>%
    select(-`Date/Time`) %>%
    rename(power = everything()) %>%
    mutate(power = power + rnorm(length(power), sd = 0.05 * sd(power))) %>%
    mutate(power = case_when(power < 0.0 ~ 0.0, TRUE ~ power))
################################################################################

# load library
library(epluspar)

# create a `BayesCalibJob` object:
bc <- bayes_job(idf, path_weather)

# specify parameters that can be measured
bc$input("VAV_1 Supply Equipment Outlet Node", "System Node Mass Flow Rate", "Hourly")

# specify the parameter to predict
bc$output("VAV_1_Fan", "Fan Electric Power", "Hourly")

# specify parameters to calibrate
bc$param(
    VAV_1_Fan = list(fan_total_efficiency = c(0.4, 0.8)),
    .num_sim = 30, .names = "FanEfficiency"
)

# get sample parameter values generated using Latin Hypercube Sampling (LHS)
bc$samples()

# run simulations from Jul 1st to Jul 3rd
bc$eplus_run(run_period = list("example", 7, 1, NULL, 7, 3))

# gather simulated data in 6-hour time frequency
bc$data_sim("6 hour")

# set field data
bc$data_field(fan_power)

# get input data for Stan
bc$data_bc()

# run Bayesian calibration using Stan
res <- bc$stan_run(iter = 50, chains = 2)
```

```r

# extract posterior distributions of calibration parameter
dist <- bc$post_dist()

# extract prediction values
pred <- bc$prediction()

# evaluate the uncertainties including NMBE and CV(RMSE)
uncert <- bc$evaluate()

# draw a density plot for the posterior distributions of calibration parameters
p_dist <- dist %>%
    pivot_longer(-sample) %>%
    ggplot() +
    geom_density(aes(value, fill = name), alpha = 0.5) +
    geom_vline(aes(xintercept = mean(value)), linetype = 2, size = 1)

# draw a boxplot to show the distributions of uncertainty indices
p_uncert <- uncert %>%
    pivot_longer(-sample) %>%
    ggplot() +
    geom_boxplot(aes(name, value, fill = name), alpha = 0.5)
```

Listing 4: Bayesian calibration