Mortar: An Open Testbed for Portable Building Analytics

Gabe Fierro
UC Berkeley
gtfierro@cs.berkeley.edu

Marco Pritoni
Lawrence Berkeley National Laboratory
mpritoni@lbl.gov

Moustafa AbdelBaky
UC Berkeley
moustafa@cs.berkeley.edu

Paul Raftery
UC Berkeley
p.raftery@berkeley.edu

Therese Peffer
UC Berkeley
tpeffer@berkeley.edu

Greg Thomson
UC Berkeley
thomsong@berkeley.edu

David E. Culler
UC Berkeley
culler@cs.berkeley.edu

ABSTRACT

Access to large amounts of real-world data has long been a barrier to the development and evaluation of analytics applications for the built environment. Open data sets exist, but they are limited in their span (how much data is available) and context (what kind of data is available and how it is described). Evaluation of such analytics is also limited by how the analytics themselves are implemented, often using hard-coded names of building components, points and locations, or unique input data formats.

To advance the methodology for how such analytics are implemented and evaluated, we present Mortar: an open testbed for portable building analytics, currently spanning 90 buildings and containing over 9.1 billion data points. All buildings in the testbed are described using Brick, a recently developed metadata schema, providing rich functional descriptions of building assets and subsystems. We also propose a simple architecture for writing portable analytics applications that are robust to the diversity of buildings and can configure themselves based on context. We demonstrate the utility of Mortar by implementing 11 applications from the literature.

CCS CONCEPTS

• **Information systems** → *Graph-based database models*; *Information retrieval*; • **Computing methodologies** → *Model development and analysis*;

KEYWORDS

Smart Buildings, Modeling and Analytics, Data Set

ACM Reference Format:
Gabe Fierro, Marco Pritoni, Moustafa AbdelBaky, Paul Raftery, Therese Peffer, Greg Thomson, and David E. Culler. 2018. Mortar: An Open Testbed for Portable Building Analytics. In *The 5th ACM International Conference on Systems for Built Environments (BuildSys '18), November 7–8, 2018, Shenzen, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3276774.3276796

1 INTRODUCTION

Building analytics applications vary in complexity from simple algorithms that only require a few directly-related data streams to algorithms using complex white- or grey-box models, requiring detailed information about the building. This information typically must be acquired from diverse sources, such as from a building automation system, architectural and mechanical drawings, operations and maintenance documents, and human input from the building operator. Implementations of analytics applications are often done as "one-offs" because of the high degree of site-specific application logic. For example, this logic can be based on the types of subsystems in the building, the names of BMS points needed for the application, and how those BMS points relate to spatial elements of the building. As a result, many analytics algorithms go largely untested and unevaluated beyond the handful of buildings (or simulations) the algorithm was developed against.

This pattern produces implementations that suffer from bias and have little generalizability [15]. A recent evaluation of U.S. building energy benchmarking and transparency programs [24] found that "indications of [energy] savings should be considered preliminary... because of the limited period of analyses and inconsistencies among analysis methods for the various studies." This is a lost opportunity for robust evaluations of analytical applications.

The existence of a large collection of building data would grant data scientists, algorithm developers, and building managers the opportunity to empirically evaluate their work on a more diverse body of buildings. Standard, open data sets and workloads exist in many other areas (such as TPC-C [30] for relational databases and MNIST [19] for digit recognition). There are several existing and prior efforts working to provide such an open data set for buildings [6, 7, 12, 18, 23, 33]. However, these data sets are limited in their span (how much data is available) and context (what kind of data is available and how it is described), which limits their efficacy for the evaluation of analytics applications.

In addition to the need for comprehensive, up-to-date, and well-annotated data sets, we also need to make applications portable in order to minimize the effort in running analytics applications against different buildings. Portable applications can nimbly provide retrocommissioning, fault detection, and diagnostics across a wide range of buildings with minimal reconfiguration.

The goal of this work is to address these challenges by providing an open testbed and a platform for storing, describing, updating, discovering, and retrieving building data. The testbed currently spans 90 buildings and contains over 9.1 billion data points. We utilize and extend Brick [5], a recently developed metadata schema that provides rich functional descriptions of building assets and subsystems, to describe not only data streams but the mechanical, electrical, logical, and functional context of those streams within all buildings in the testbed. Using this standardized and extensible method of describing buildings, we show that it is possible to create a data platform that can subsume the heterogeneous data produced by the built environment. Further, we also present an architecture for portable and extensible analytics applications that leverages Brick to simplify the development and evaluation of these applications across multiple buildings.

Together, the testbed and the portable architecture provide a platform, named Mortar, which lowers the integration cost of deploying new analytics applications and acts as a vehicle for reproducible evaluations. We evaluate Mortar by implementing a representative set of analytics applications; these implementations as well as the data sets and Brick models that are part of the testbed will be released for others to leverage and build upon.

The proposed platform will serve as a repository of open-source, vetted, and robust implementations of building analytics algorithms. We intend for this platform to be used by algorithm developers and researchers for the implementation and evaluation of analytics applications, and by building managers who upload and describe their building data in order to run any available analytics to manage their portfolio. The data, Brick models and libraries mentioned in this paper are available as part of the Mortar platform (available online at mortardata.org with creation of a free account); the platform will soon support user-provided datasets and metadata models.

The contributions of this paper are:

(1) the design and implementation of a platform for the development and evaluation of portable building analytics
(2) a modular and extensible architecture for portable analytics applications
(3) a data set of 90 buildings containing over 9.1 billion data points, where the relationships between data points are well described and annotated using Brick models

The remainder of the paper is organized as follows. §2 provides a brief background on Brick. §3 details the design and implementation of Mortar. §4 presents the architecture of portable applications. A detailed description of the building data sets is presented in §5, followed by the evaluation of Mortar in §6. Related work is presented in §7 and the paper concludes in §8 outlining future work.

2 BRICK

Brick [5] is an open-source ontology providing a unified semantic representation of building assets, subsystems, and the relationships between them. Brick has two components: an extensible *class hierarchy* representing the physical and logical entities in buildings, and a minimal set of *relationships* that capture the connections between entities. A Brick model of a building is a labeled, directed graph in which the nodes are entities and the edges are relationships.

2.1 Why Brick?

Brick has several advantages over alternative metadata representations such as Project Haystack [1] and IFC [8].

(1) Completeness: Brick's class hierarchy can describe the diverse array of equipment, points, subsystems, and other physical or logical assets found in buildings. The class hierarchy can be extended to define new or unusual entities.
(2) Flexibility: by utilizing the class hierarchy and transitive relationships in queries against Brick models, applications can remain agnostic to the level of detail of a Brick model. This allows applications to run on buildings whose Brick model may not be fully specified, but contains the points and relationships necessary for the application to run.
(3) Consistency: the minimal set of relationships defined by Brick and the constraints defined by the Brick ontology help reduce variability in how the same concepts are expressed across independent Brick models.
(4) Queryability: applications use the powerful SPARQL [14] language to query Brick models. SPARQL enables applications to traverse the complex and heterogeneous graph structures that typify Brick models and retrieve the information they need to operate. [14] provides a full explanation of the SPARQL language and [13] contains a discussion of the application of SPARQL to Brick.

These properties make Brick an ideal candidate for addressing the metadata-related shortcomings in existing platforms. Further, Brick lays the foundation for developing and evaluating portable analytics applications, which is described in details in §4.

2.2 Brick Extensions

Although the current 1.0.3 release of the Brick schema can describe all of the equipment and points contained in the testbed, it does not contain site-level properties such as building typology that are helpful for qualifying sites. Table 1 lists the properties we have added to Brick; these changes are being collected into a formal proposal for inclusion into an upcoming release of Brick.

3 MORTAR

Mortar stands for **M**odular **O**pen **R**eproducible **T**estbed for **A**nalysis & **R**esearch. It provides a platform for the development and evaluation of portable building analytics applications. Mortar contains timeseries data consisting of historical values of the sensors, actuators, setpoints and other data points for a large number of buildings. Mortar manages a Brick model for each of these buildings which describes the properties of the building as well as the equipment, sensors, subsystems and relationships within that building. Mortar also includes a library of analytics applications, structured according to the architecture detailed in Section 4. These applications access timeseries data by executing queries against the Brick model to refer

Property	Definition
`hasSite/isSiteOf`	`brick:Site` entity to capture points belonging to a single facility
`noaa`	Nearest NOAA weather stations. Gives rough location of site
`area`	Floor area (typically m^2 or ft^2). Can be associated with `brick:Site` (for total area) or `brick:Floor`, `brick:Room`, etc for more fine-grained annotation
`adjacentTo`	Adjacency for rooms, other spaces
`uuid`	timeseries identifier used by BTrDB
`hasUnit`	engineering units for the associated timeseries

Table 1: Proposed set of Brick extensions for capturing building properties

to relevant data streams. This methodology enables portable analytics applications, made more robust through evaluation against the diverse population of buildings in the testbed.

3.1 System Architecture

The architecture of Mortar is driven by the need to a) link historical timeseries data with its context as captured by a Brick model, b) execute code that retrieves segments of timeseries data filtered by context, and c) provide those features at the scale of tens or potentially hundreds of buildings. Mortar is assembled from a family of open-source software. Mortar has four components as shown in Figure 1: timeseries store, Brick model storage, query processor, and application execution environment.

Timeseries storage: one of the critical features of the testbed is its support for progressively updated data sets ("live" data). Building managers and other users need to be able to upload recent data for existing building points as well as upload historical data for newly annotated building points.

The testbed manages the distribution and updating of data using a full timeseries database instead of distributing data in static file form (such as CSVs). At the testbed's current scale of over 145 GB of 64-bit floating-point values and 64-bit nanosecond-precision timestamps, it is unreasonable to expect users to download the data themselves. Many timeseries databases offer filtering and aggregation features that allow applications to specify and download just the data they need for their execution. The process of appending new data to an existing stream is also made much easier through the use of a timeseries database.

Brick model storage: another key aspect of the architecture is the storage of Brick models over time, and the integration of the Brick model with the timeseries database. Mortar needs to store a Brick model for each building and maintain the history of those models as they are changed and updated. Brick models use the `brickframe:uuid` property to tag a point in Brick with the stream identifier for its historical data in the timeseries database. The Brick model decouples the context and name of a point from its timeseries representation, which opens the possibility for the testbed to change the timeseries representation without affecting how applications are run. This is helpful for maintaining continuity of data collection through repairs and retrofits, processing data (e.g., for re-uploading

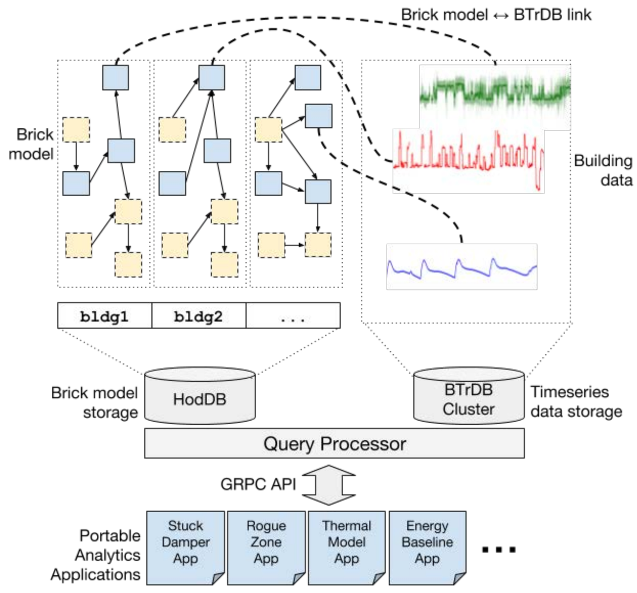

Figure 1: Mortar architecture

data with an adjusted calibration parameter), and in cases of fixing errors (e.g., if the wrong data was uploaded for a building point). In this way, Mortar supports the natural evolution of buildings. Because the qualification and execution of applications makes heavy use of Brick queries, it is important that the storage medium for the Brick model maintains low query response times despite a large number of concurrent requests against a large number of buildings.

Query processor: the query processor exposes to applications a unified interface for accessing both the timeseries database and the Brick model storage. The query processor understands the `brickframe:uuid` property binding the timeseries and Brick model databases, and performs any caching and additional processing of the query not performed by the timeseries or Brick model databases. The interface itself is described in-depth in Section 4.2.

Application Execution Environment: Mortar's query processor presents a network-facing API, so applications may be run either in Mortar's execution environment or locally on a client's computer. Placing the execution environment within Mortar reduces the amount of data and computation that needs to be done on a client's computer; development and execution of an application can happen in a web-based frontend such as Jupyter Lab [16].

3.2 Implementation

Mortar is currently deployed on a small Kubernetes-managed cluster; it contains Brick models for 90 buildings, over 9.1 billion data points, and 11 applications.

Timeseries database: the testbed uses the BTrDB timeseries database [4], which provides fast storage and retrieval of scalar-valued timeseries data. BTrDB supports adding storage capacity incrementally using the Ceph storage engine, allowing the testbed to scale gracefully as the number of buildings and amount of data in the testbed increases. BTrDB supports querying data at arbitrary resolutions, grouping data using associative statistical aggregates `min`, `mean`, `max` and `count`, as well as querying the raw data itself.

Brick model database: existing open-source RDF/SPARQL databases do not perform well under a Brick model workload, often taking seconds or even minutes to execute a single query on a moderately-sized Brick model [13]. Thus, Mortar stores all Brick models in HodDB [13], a Brick-specialized RDF/SPARQL database and query processor that demonstrates good performance on the testbed workload.

Query processor: the query processor[1] is implemented in the Go programming language and interacts directly with BTrDB and HodDB, which are deployed on the same cluster. When the query processor receives a request from a client, it evaluates the included Brick queries to resolve the identifiers for timeseries streams in BTDB, and then pulls the data from BTrDB at the desired resolution for the desired segment of time. The query processor can simplify the data cleaning process for the client by performing unit conversion (e.g., converting all temperature data to Celsius) and aligning timestamps of returned data (e.g., aligning hour-bucket aggregated data to the top of the hour).

Application execution environment: clients interact with the testbed through the query processor, which presents a network-facing API implemented using GRPC. GRPC[2] is an open-source RPC framework with several key features:

(1) streaming data: GRPC makes it easy to stream large amounts of data over a connection
(2) session management: GRPC supports cancellation of long-running operations (such as fetching historical data) which can help recover server resources if a client abandons a request
(3) autogenerated client bindings: GRPC enables clients written in most major programming languages to interact with the testbed. Currently we have only generated bindings for the Python programming language.

The query processor delivers all timeseries data to the client encoded using Apache Arrow[3], an efficient in-memory format for columnar data designed to be language-independent. Apache Arrow will soon support zero-copy semantics, which, coupled with Mortar's on-server application execution environment, should improve the performance of client applications. Additionally, the use of Apache Arrow opens the possibility of integrating Mortar with open-source projects for data science and machine learning such as Pandas [22], Hadoop [32], and TensorFlow [2].

4 ANATOMY OF PORTABLE APPLICATIONS

Portability measures how easily a program can be moved from one computing environment to another. In the context of building analytics, portability refers to how well a piece of code generalizes to multiple heterogeneous buildings. Many available implementations of building analytics are not portable due to hardcoded point names, assumptions about the structure of a building and its subsystems, assumptions about the availability of data, and tightly-coupled phases of operation. These limitations are an inevitable consequence of how few real buildings are available for the development or evaluation of analytics applications, which makes it difficult to perform a robust evaluation.

In this section we describe the general structure of a portable building analytics application designed to be executed against potentially hundreds of buildings with very little configuration. We decompose all Mortar applications into five components: `qualify`, `fetch`, `clean`, `analyze`, and `aggregate` (see Figure 2). These are executed in order by the platform when the application is invoked.

4.1 Application Requirements

The **`qualify` component** defines the metadata and data requirements of an application. Mortar evaluates these requirements against all available buildings in order to determine the subset of buildings against which the application can run (the *execution set*). The ratio between the execution set and the total number of buildings in the testbed provides a good measure of the portability of an application, and can be used iteratively, to allow an application to cover more buildings. Mortar forks an instance of the application for each site in the execution set. Specifically, the `qualify` component checks

(1) constraints on building typology and other properties, such as the number of floors in a building, floor area, climate, and occupancy class
(2) data context constraints, such as the kinds of equipment in the building and available relationships
(3) data availability constraints, including the amount of historical data and available data resolution

The first two constraints are defined using Brick queries. When qualifying an application, Mortar evaluates these queries against the Brick model for each site in the testbed; sites are placed in the execution set if the Brick queries return results. An application can use a set of Brick queries to implement a "decision tree" where the application customizes its execution based on the information available in the Brick model, such as to distinguish between RTU-based and AHU-based HVAC systems. Mortar evaluates data availability constraints against the data streams identified by the Brick queries.

An example `qualify` specification for the "Rogue Zone Detection[4]" application is shown in Figure 3. The Brick query looks for terminal units that expose air flow setpoints and sensors and relates the terminal unit to the HVAC zone it conditions. Brick's subclass relationships allow the same query to be used to find systems with a single air flow setpoint and those with a high/low setpoint deadband. In these cases where there is variability in the type of system or what data is exposed, it is up to the application developer how to deal with the difference: the developer can write more precise Brick queries to simplify the application code to only deal with a single type of system, or the developer can account for the potential differences thereby covering more buildings.

4.2 Data Retrieval

Mortar runs the **`fetch` component** for each site in the execution set. The `fetch` component performs the actual retrieval of data from the timeseries database corresponding to the set of streams

[1]We developed an improved version of the MDAL service [13], which is used in the query processor mechanism.
[2]https://grpc.io/
[3]https://arrow.apache.org

[4]This application is described in more details in Section 6.1.

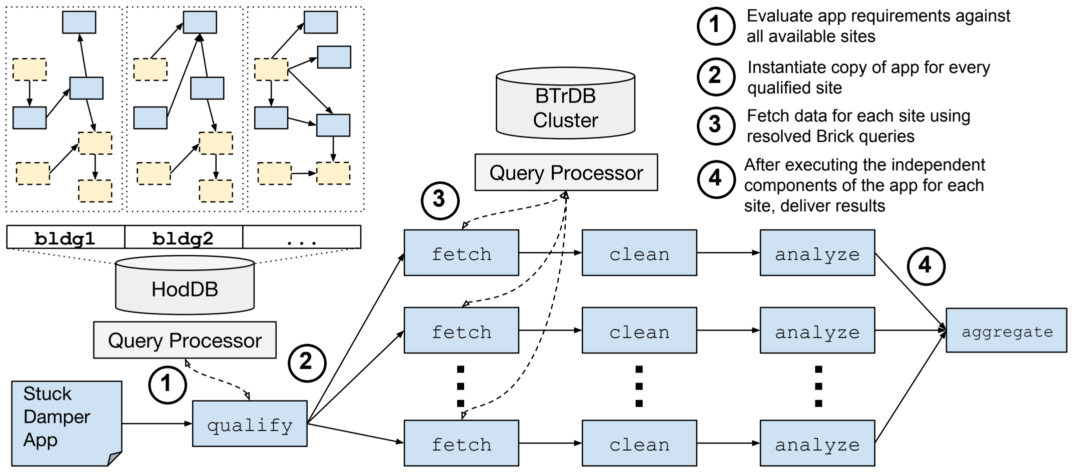
1 Evaluate app requirements against all available sites
2 Instantiate copy of app for every qualified site
3 Fetch data for each site using resolved Brick queries
4 After executing the independent components of the app for each site, deliver results
BTrDB Cluster
Query Processor
bldg1
bldg2
...
HodDB
Query Processor
Stuck Damper App
qualify
fetch
clean
analyze
aggregate

Figure 2: Architecture of a portable analytics application. An application consists of five segments: `qualify` which filters the building corpus into the execution set, `fetch`, `clean` and `analyze` which are all executed once for each building in the execution set, and `aggregate` which is executed once over all outputs of `analyze`.

```
queries:
  required:
   - >
     SELECT ?vav ?sen ?sp ?sen_id ?sp_id FROM %s WHERE {
       ?sp  rdf:type/rdfs:subClassOf* brick:Air_Flow_Setpoint .
       ?sen rdf:type/rdfs:subClassOf* brick:Air_Flow_Sensor .
       ?equip rdf:type/rdfs:subClassOf* brick:Terminal_Unit .
       ?sp  bf:isPointOf ?equip .
       ?sen bf:isPointOf ?equip .
       ?sp  bf:uuid  ?sp_id .
       ?sen bf:uuid  ?sen_id .
       ?equip bf:feeds+ ?zone .
       ?zone rdf:type brick:HVAC_Zone .
     };
```

Figure 3: YAML app specification for qualifying sites for "Rogue Zone (Airflow)" application

identified by the Brick queries. The data retrieval request uses the following parameters:

(1) "variable" definitions: these map a name to a Brick query defining the context for a point and the desired engineering units for that point (if known), and aggregation function (min,max,mean,count, or raw).
(2) temporal parameters: defines the bounds on the data, desired resolution, and if we want aligned timestamps.

The output of the `fetch` component is an object providing access to the results of the Brick queries, the resulting timeseries dataframes, and convenience methods for relating specific dataframes based on the Brick context (for example, the setpoint timeseries related to a given sensor timeseries).

Figure 4 has an example of the `fetch` component for the "Rogue Zone Detection" application. The query requests 11 days of data on air flow sensors and setpoints aggregated by mean in 10-minute buckets. The `Aligned` flag is true, which means the returned timeseries data will have the same time base.

```
def run(site):
  print 'running', site
  data = {}
  resp = client.query({
      "Composition": ["airflow","setpoint"],
      "Aggregation": {
          "airflow": ["MEAN"],
          "setpoint": ["MEAN"],
      },
      "Variables": {
        "airflow": {
          "Definition": """SELECT ?vav ?sen ?sen_id FROM %s WHERE {
            ?sen rdf:type/rdfs:subClassOf* brick:Air_Flow_Sensor .
            ?sen bf:isPointOf ?equip .
            ?sen bf:uuid  ?sen_id .
            };""" % site,
           "Units": units.CFM
          },
          "setpoint": {
            "Definition": """SELECT ?vav ?sp ?sp_id FROM %s WHERE {
              ?sp rdf:type/rdfs:subClassOf* brick:Air_Flow_Setpoint .
              ?sp bf:isPointOf ?equip .
              ?sp bf:uuid ?sp_id .
            };""" % site,
            "Units": units.CFM
          },
      },
      "Time": {
          "Start":  "2015-05-01T10:00:00-07:00",
          "End":  "2015-05-12T10:00:00-07:00",
          "Window": '10m',
          "Aligned": True,
      },
  })
  obj['obj'] = resp.serialized
  obj['data'] = resp.df
  yield obj_store.put(obj)
```

Figure 4: Python `fetch` step for "Rogue Zone (Airflow)" application

4.3 Data Cleaning

The **`clean` component** is executed on the output of the `fetch` component. The purpose of this component is to normalize the timeseries data for the `analyze` component, which is executed next. Common operations in the `clean` component are hole filling,

```python
def run(objid):
    data = obj_store.get(objid)
    data['cleaned'] = data['data'].dropna() # drop null values
    return obj_store.put(data)
```

Figure 5: Simple Python `clean` step to drop null values (periods of missing data)

```python
def run(objid):
    data = obj_store.get(objid)
    resp = deserialize(data['obj'])
    df = data['data']
    # loop through data columns
    for sensor in resp.mapping['airflow']:
        # find the setpoint corresponding to this sensor
        setpoint = resp.find(sensor, join_on='?vav', extract='?sp_id')
        dd = df[[sensor, setpoint]]
        dd.columns = ['airflow','setpoint']
        if len(dd[dd.airflow < dd.setpoint]):
            bad = dd.airflow < dd.setpoint
            # get runs of airflow being below setpoint
            dd['same'] = bad.astype(int).diff(1).cumsum()
            groups = dd.groupby('same').groups
            for key, grp in groups.items():
                print 'VAV {0} had high airflow from {1} to {2}'
                  .format(resp.context[sensor]['?vav'], grp[0], grp[-1])
```

Figure 6: Python `analyze` step to find periods when the measured airflow is lower than the setpoint in the Rogue Zone Airflow application

specialized aggregation, and data filtering. It is kept modular to facilitate the re-use of standard cleaning steps. Application developers can build their own cleaning components or leverage existing methods. A simple example of a `clean` component that drops missing data is presented in Figure 5.

4.4 Application Execution

The **`analyze` component** contains the actual application logic and is also executed for each site in the execution set. It can optionally output a result that is delivered to the final component. Figure 6 contains the logic of the “Rogue Zone Detection” application. It finds contiguous segments of time during which the measured airflow is lower than the airflow setpoint.

The **`aggregate` component** is an optional component that is executed once for an application, and takes as an argument the output of the `analyze` component from all sites. This is where an application can perform any final aggregation or analysis across sites.

5 THE MORTAR DATASET

As of writing, Mortar contains 9.1 billion data points of timeseries data for 90 buildings, constituting more than 750 million hours of data. The majority of data streams in Mortar are at a 15-minute interval, though some are more fine-grained (up to 1-second).

Figure 7 describes the distribution of the number of streams per building. Each building is accompanied by a Brick model that describes the building, its equipment and subsystems, available points, and references to timeseries data streams. The Brick models in the Mortar data set range in size from 2,117 to 8,763 nodes. Table 2 enumerates some of the types of available points and equipment in the testbed.

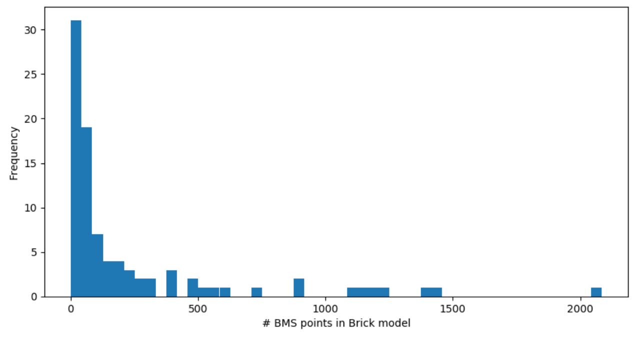

Figure 7: Histogram of number of data streams for all sites (μ =241).

Temperature Sensor	7380	Luminance Sensor	257
Occupancy Sensor	445	Pressure Sensor	148
Outside Air Temp. Sensor	362	Cloud Cover	32
Setpoints (generic)	2331	Power Meters	77
VAVs	4724	AHUs	467
HVAC Zones	4887	Dampers	1662
Non BMS Thermostats	123		

Table 2: Count of streams and equipment available in the testbed data set, aggregated by type. AHU and VAV totals include related equipment such as fans and pumps.

Depending on what data is available, Brick models are generated from Building Information Modeling (BIM) models of buildings, architectural diagrams, site visits, and interviews. The framework for generating Brick models is open-source[5], and aims to integrate with a more mature Brick model generation framework such as Scrabble [17].

The majority of the dataset is made up of large commercial buildings belonging to a university campus. The buildings are typically used as offices, classrooms, research facilities and health care clinics. The average building has a floor area of 70,000 sqft, 3 floors and more than 100 rooms, while the largest building is a large library with a floor area above 400,000 sqft. Most buildings are conditioned using large built-up HVAC systems with air handlers and local distribution boxes, controlled by building automation systems. Chilled water and hot water are produced by a central plant and distributed through large pipes to most buildings. Some additional chillers are installed in some buildings to complement the central system.

Other buildings in the data set come from a set of independent data collecting efforts. Most of the non-campus buildings are part of an ongoing project to develop a building operating system; these are mostly small commercial buildings ranging from movie theatres to fire stations to animal shelters. Data collected includes thermostats, building meters, occupancy, temperature, illumination and humidity sensors, electric vehicle charging stations and solar panels.

An important concern is the anonymization of the data in the testbed. Mortar anonymizes the names and locations of buildings (using nearby NOAA weather stations to indicate general location)

[5]https://github.com/gtfierro/BrickMason

and the names of entities in the Brick metadata model. Future releases of Mortar will support the selective deanonymization of this metadata to authorized users. Mortar does not currently anonymize the timeseries data values; because Mortar supports continuous data collection, it is difficult to apply existing best-practices such as differential privacy.

6 EVALUATION

The key contribution of this paper is a testbed for the robust implementation and evaluation of portable building analytics applications. In order to evaluate the efficacy of Mortar, we have implemented a family of 11 applications. We chose these applications to exercise the ability to pull data from buildings using queries against a Brick model, and also to make use of the proposed architecture for portable applications. To test the former, we implement applications that require the use of Brick relationships to properly describe the data needed to run the application. To test the latter, we implement all applications using the decomposed architecture proposed in Section 4; some of the applications involve similar or identical logic and can re-use each other's components, using Brick queries to adjust the source data.

6.1 Applications

In this section we provide a brief description of the applications that were used to evaluate Mortar. All applications are implemented in the Python programming language, which allows application developers to make use of the mature set of data science and statistics libraries available. A summary of the implementations of these applications is provided in Table 3. The portable implementations of these applications will be released in tandem with the data set, forming the beginning of an open library of analytics applications.

1. Baseline Calculation: this application ports an existing open-source package (LBNL-baseline[6]), which implements a baseline calculation algorithm [21], to Mortar. The package requires access to electric load data as well as outside temperature data when available, which is retrieved using a simple Brick query. Porting this application to Mortar allows us to easily compare a predicted baseline with historical data across a variety of sites with differing climates, construction, and usage patterns. Some results of this application are in Figure 10.

2. Baseline Deviation: we made use of the modular implementation of the Baseline Calculation application to implement this application. In particular, we modified the `main` component from the Baseline Calculation application (keeping the rest of the stages) to compare measured energy consumption with the predicted baseline to identify periods of abnormally high consumption.

3. Energy Usage Intensity (EUI) Calculation: the calculation of EUI involves dividing the yearly energy consumption of a building in kilowatt-hours by the total floor area of that building. This metric is often used to benchmark buildings. Our implementation of EUI uses a Brick query to discover what kind of building energy consumption data is available. In the current data set, sites report consumption either through an instantaneous demand meter such as a Rainforest Eagle[7] or through an energy bill such as PG&E's "Share My Data" service. The floor area of a building can also be captured in the Brick model using the extensions from Table 1. The logic of the application retrieves a year of data, performs the appropriate calculation (either a conversion from instantaneous power data to kWh or a simple sum of regular kWh readings) and divides it by the total floor area retrieved from the Brick model.

4. HVAC Energy Disaggregation: this application estimates the energy consumption of equipment by correlating changes in electrical demand (through the use of a building-level meter or submeter) with changes in HVAC equipment state. In the case of binary state points (e.g. `Compressor On Off Status`), the disaggregation application can estimate the total energy usage of a piece of equipment. The disaggregation application could then make use of any available weather and internal temperature data in order to identify any change in efficiency in the equipment's operation corresponding to changes of environmental variables. Our implementation leverages the Brick class hierarchy to find all points of type `Status` that relate to equipment in the building; the resulting queries can easily be refined to look only at specific classes or instances of equipment such as only single-stage or multi-stage equipment.

5. Thermal Model Identification: this application trains a zone-level thermal model using zone- and room-level (using additional sensors, if present) temperature sensing, outside air temperature, cloud coverage, HVAC equipment state data and, when available, adjacency information. The implementation uses Brick queries to determine what information is available such as the number of zones, the temperature sensing in those zones, HVAC equipment conditioning those zones, and available weather data.

6. Rogue/Critical Zone Temperature: this application detects rogue zones, which are thermal zones whose temperature consistently measures outside of the setpoint temperature band; this can be caused by thermal loads larger than design conditions, incorrect setpoints and/or broken sensors or equipment [9]. For critical zones, it indicates that the air temperature leaving the AHU should be lower/higher. In large HVAC systems these zones can cause an increase in the energy use of the entire air handling units, due to increased reheat in the other zones. The detection of rogue zones can also be used to improve control sequences.

7. Rogue/Critical Zone Airflow: this application detects rogue zones whose airflow is consistently below the airflow setpoint. It uses Brick models to pull in additional context about the building. The airflow setpoints and sensors can be related through the HVAC infrastructure to the zones containing the rooms affected by the reduced airflow. This information is typically used as an input to the duct static pressure controller on the air handling unit serving these zones.

8. Stuck Damper Detection: this application performs the common fault-detection task of identifying dampers whose position has not changed in weeks or months. Our implementation automates the task of finding the damper position status streams and identifies the HVAC zones and rooms affected by the damper. For zones with discharge airflow sensors the application can also test if the damper appears to work correctly, but the amount of air supplied does not change with its position. This can indicate a faulty sensor or a broken linkage between actuator and the physical damper.

[6]https://github.com/LBNL-ETA/loadshape
[7]https://rainforestautomation.com/rfa-z114-eagle-200/

Category	Application	Brick LOC	App LOC	# buildings	% coverage
Measurement, Verification & Baselining	Baseline Calculation	3	120	33	37%
	EUI Calculation	10	100	7	8%
	HVAC Energy Disaggregation	14	124	14	16%
	Thermal Model Identification	17	339	17	19%
Fault Detection & Diagnosis	Rogue Zone Temperature	15	104	56	62%
	Rogue Zone Airflow	7	98	3	3%
	Baseline Deviation	3	204	14	16%
	Stuck Damper Detection	8	91	30	33%
	Obscured/Broken Lighting Detection	11	100	2	2%
Advanced Sensing	Virtual Coil Meter	14	150	60	67%
	Chilled Water Loop Total Electrical Consumption	17	160	15	17%

Table 3: Applications: Brick LOC and App LOC indicate the lines of code needed to define the Brick queries and application logic, respectively. "% coverage" is what proportion of the testbed's buildings qualified for that application; the corresponding number of buildings is in the "# buildings" column.

9. Obscured Lighting Detection: this application groups lighting status points for a lighting zone with luminance sensors in that zone in order to build a correlation between time, lighting status, and luminance. Deviations from this model can be used to identify broken or obscured fixtures and luminaires.

10. Virtual Coil Meter this application estimates the amount of heat energy used by a heating or cooling coil by performing a calculation over upstream and downstream air temperature sensors, air flow sensors, and the position of the valve in the coil [28]. The portable implementation of this application discovers these points through querying the Brick model for a building for their relationships to each other and the building's HVAC system.

11. Chilled Water Loop Total Electrical Consumption: this application examines all equipment on a chilled water loop and sums the electrical consumption of any component found on that loop. Assembling this collection of equipment and related building points without a Brick model involves a high degree of manual effort that does not carry over when porting the application to other buildings.

6.2 Results & Discussion

The open building testbed presents an opportunity to compare multiple buildings in a portfolio, measure the portability of an application, measure the accuracy or performance of an application, compare the performance of similar applications, and automate tedious re-configuration when porting an application to different buildings. To demonstrate these features, we examine the results of running a few applications from Table 3 against the Mortar testbed.

First, we examine the accuracy of the building energy estimation baseline algorithm from [21] across winter and summer seasons (Figure 9). In winter, the mean squared error of the baseline prediction compared to a week's ground truth increases with the floor area of the building. Because the positive correlation between square footage of the building and the error in energy prediction suggests that larger buildings have more variability in their loads than the baseline estimation algorithm is able to account for; the higher error in the winter months compared to summer months additionally suggests a seasonality to this error. A more thorough evaluation of the baseline estimation algorithm is beyond the scope of this paper, but this initial result demonstrates the value of being able to evaluate an analytics algorithm over a large corpus of buildings.

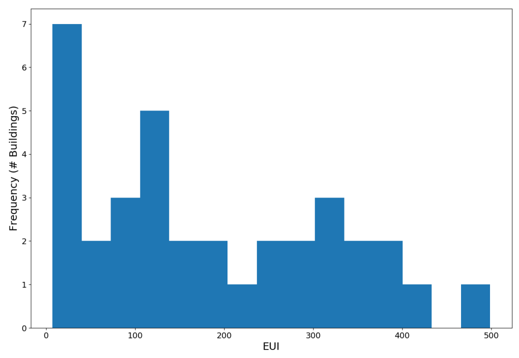

Figure 8: Distribution of EUI across the Mortar portfolio

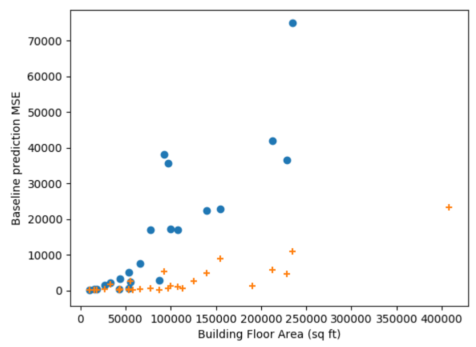

Figure 9: Comparing the mean squared error of building energy consumption prediction using the baseline from [21] trained and evaluated on summer months (+) versus winter months (●).

The Thermal Model application is a good example of how applications can change their behavior based on the information available in a building. The `qualify` step looks for buildings with thermostats and RTUs with or without room-level temperature sensors. In the case when only thermostats are available, the application can provide a zone-level thermal model that runs on 17

Name	Year	Available Data	Updates	Metadata
Building Genome Project [23]	2017	non-residential building electric meter data	rare	building typology
CBECS [12]	2016	building energy usage, systems and equipment survey	4-10 year cycle	building typology
NILMTK [7]	2014	high-frequency building electric meter data, equipment state	rare	labels and electrical subsystem
Smart* [6]	2012	electrical meter data, temperature and humidity sensors, applicance state for residential buildings	yearly releases	labels only
REDD [18]	2011	full building and device-level electrical meter data	none	electrical subsystem
BASE [33]	1995	temperature and humidity sensors, HVAC equipment and sensors	none	building typology, HVAC system
GREEND [25]	2013	device-level electrical meter	none	appliance labels
Pecan St Dataport [27]	2011-2018	building and appliance level electric, gas, and water data for residential buildings	unspecified	building typology, equipment labels
REFIT [26]	2013-2015	high frequency electric building and appliance level meter data	none	building typology, appliance labels

Table 4: Some open data sets of building data. "Labels only" means the data set's metadata only identifies a timeseries with a label and little or no contextual metadata. "Building typology" means the data set's metadata only deals with high-level information about the building such as occupancy class, climate and floor area.

buildings from the testbed. For buildings where room-level temperature sensors are available, the application takes those into account to provide a more fine-grained room-level thermal model, but this only runs on 3 buildings from the testbed. If the application were to be extended with a thermal modeling approach for AHU-based systems with heating and cooling coils, then the thermal model application would operate over all 90 buildings.

The Mortar testbed also presents the opportunity to examine building performance across a portfolio. In Figure 8, we plot the distribution of EUI over the whole testbed portfolio; users could adjust the Energy Use Intensity application `qualify` step to filter by different building properties, such as climate and square footage. The inclusion of building properties in the Brick model for each building facilitates this flexible portfolio management.

Mortar also lets users skip tedious configuration steps. Without the use of a Brick model, assembling the points for the Chilled Water Loop Total Electrical Consumption application requires site-specific knowledge as to what power meters are present in the building, what equipment they measure, and whether or not that piece of equipment is related to a particular chilled water loop. Performing this assemblage over a large building portfolio can be unmanageable.

7 RELATED WORK

Existing building benchmarking data sets (Table 4) are static, sparse, and/or lack sufficient context to implement potentially complex analytics. Further, there is no standard adopted naming scheme to tie these data sets together, which hinders the portability of any application developed against a single platform. In contrast, Mortar provides data sets that are continuously updated, and well annotated. Further, many of the existing data sets can be easily integrated in Mortar by representing their metadata using a Brick model and ingesting any included timeseries data. Executing and automating this process is the subject of future work.

Also related are a family of analytics platforms for building data and the smart grid. Commercial platforms like Skyspark [29] and

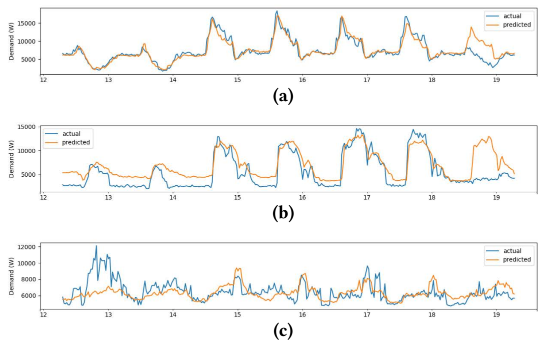

Figure 10: Baseline prediction app: Predicted baseline using [21] plotted against actual building energy consumption.

BuildingOS [20] offer a mixture of pre-built standard analytics and composable rules and operators for user-defined analytics. Some of these platforms require analytics to be implemented in a proprietary or custom programming language, which limits the types of tools that can be used to develop analytics. There are also several academic building operating systems that implement building and smart-grid analytics [3, 10, 11, 31].

The proposed portable analytics applications architecture enables users to develop applications (or port existing ones) using standard, open-source tools and frameworks, and integrate these applications into the Mortar platform. Further, Mortar allows users to add their own buildings to the testbed, and will assist them with the creation of the Brick model. Documenting and automating this process is the subject of future work.

8 CONCLUSION & FUTURE WORK

In this paper, we have presented the design and implementation of an open testbed for portable building analytics (Mortar), a diverse corpus of timeseries data and Brick models spanning multiple building classes (Mortar data set), and an architecture for implementing portable analytics applications (Mortar portable applications). Mortar utilizes and extends Brick, a graph-based metadata model, to describe the relationships between data streams and to facilitate application portability. We have developed 11 applications from the literature to evaluate Mortar. The evaluation shows that Mortar can run a single application against multiple buildings to a) compare the performance of multiple buildings in a portfolio (EUI score), b) measure and improve application coverage across buildings, and c) measure the application performance or accuracy (e.g., mean error). Further, we have showed that Mortar allows application developers to significantly reduce the manual labor involved in developing and deploying an application against multiple buildings, or compare the performance of two algorithms for a given application. Our goal is that Mortar will be used as a standard benchmarking platform for implementations of new and established analytics algorithms, leading to a new generation of robust, portable analytics with reproducible evaluations.

There are several areas of future work that will improve the usability and security of Mortar. Firstly, we are working on integrating privacy-preserving techniques for sharing continuously updated timeseries data. We are also improving the process for integrating new sources of timeseries data and metadata into the testbed. Mortar is available at mortardata.org.

9 ACKNOWLEDGEMENTS

This research is supported in part by California Energy Commission EPC-15-057, Department of Energy grant EE-0007685 and the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. The opinions expressed belong solely to the authors, and not necessarily to the authors' employers or funding agencies.

REFERENCES

[1] 2018. Project Haystack. http://project-haystack.org/. (2018).

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning.. In *OSDI*, Vol. 16. 265–283.

[3] Yuvraj Agarwal, Rajesh Gupta, D Komaki, and Thomas Weng. 2012. Buildingdepot: an extensible and distributed architecture for building data storage, access and sharing. *BuildSys '12* (2012), 64–71. https://doi.org/10.1145/2422531.2422545

[4] Michael P Andersen and David E Culler. 2016. BTrDB : Optimizing Storage System Design for Timeseries Processing. Section 3 (2016), 39–52.

[5] B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal, M. Berges, D. Culler, R. Gupta, M.B. Kjærgaard, M. Srivastava, and K. Whitehouse. 2016. Brick: Towards a unified metadata schema for buildings. In *Proceedings of the 3rd ACM Conference on Systems for Energy-Efficient Built Environments, BuildSys 2016.* https://doi.org/10.1145/2993422.2993577

[6] S. Barker, A. Mishra, D. Irwin, E. Cecchet, P. Shenoy, and J. Albrecht. 2012. Smart*: An Open Data Set and Tools for Enabling Research in Sustainable Homes. *SustKDD* August (2012), 6. https://doi.org/adf

[7] Nipun Batra, Jack Kelly, Oliver Parson, Haimonti Dutta, William Knottenbelt, Alex Rogers, Amarjeet Singh, and Mani Srivastava. 2014. NILMTK: An Open Source Toolkit for Non-intrusive Load Monitoring. (2014), 265–276. https://doi.org/10.1145/2602044.2602051 arXiv:1404.3878

[8] Vladimir Bazjanac and Drury B. Crawley. 1999. Industry Foundation Classes and Interoperable Commercial Software in Support of Design of Energy-Efficient Buildings. *5th IBPSA* April (1999), 7. http://www.ibpsa.org/

[9] Arka A. Bhattacharya, Dezhi Hong, David Culler, Jorge Ortiz, Kamin Whitehouse, and Eugene Wu. 2015. Automated Metadata Construction to Support Portable Building Applications. *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments - BuildSys '15* (2015), 3–12. https://doi.org/10.1145/2821650.2821667

[10] Bin Cheng, Salvatore Longo, Flavio Cirillo, Martin Bauer, and Ernoe Kovacs. 2015. Building a Big Data Platform for Smart Cities: Experience and Lessons from Santander. *Proceedings - 2015 IEEE International Congress on Big Data, BigData Congress 2015* December 2016 (2015), 592–599. https://doi.org/10.1109/BigDataCongress.2015.91

[11] Stephen Dawson-Haggerty, Andrew Krioukov, Jay Taneja, Sagar Karandikar, Gabe Fierro, Nikita Kitaev, and David E Culler. 2013. BOSS: Building Operating System Services.. In *NSDI*, Vol. 13. 443–458.

[12] EIA. 2016. Commercial Buildings Energy Consumption Survey (CBECS) User's Guide to the 2012 CBECS Public Use Microdata File. 2016, August (2016). https://www.eia.gov/consumption/commercial/data/2012/index.cfm?view=microdata

[13] Gabe Fierro and David E Culler. 2017. Design and Analysis of a Query Processor for Brick. *Proceedings of the 4th ACM International Conference on Systems for Energy-Efficient Built Environments* 1, 1 (2017), 11:1—-11:10. https://doi.org/10.1145/3137133.3137155

[14] Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. 2013. SPARQL 1.1 query language. *W3C recommendation* 21, 10 (2013).

[15] Eamonn Keogh and Shruti Kasetty. 2002. On the need for time series data mining benchmarks. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '02* (2002), 102. https://doi.org/10.1145/775047.775062

[16] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. 2016. Jupyter Notebooks – a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt (Eds.). IOS Press, 87 – 90.

[17] Jason Koh, Dhiman Sengupta, Julian McAuley, Rajesh Gupta, Bharathan Balaji, and Yuvraj Agarwal. 2017. Scrabble: Converting Unstructured Metadata into Brick for Many Buildings. In *Proceedings of the 4th ACM International Conference on Systems for Energy-Efficient Built Environments (BuildSys '17).* ACM, New York, NY, USA, Article 48, 2 pages. https://doi.org/10.1145/3137133.3141448

[18] J Zico Kolter and Matthew J Johnson. 2011. REDD : A Public Data Set for Energy Disaggregation Research. *SustKDD workshop* xxxxx, 1 (2011), 1–6. http://users.cis.fiu.edu/{~}lzhen001/acti

[19] Yann LeCun, Corinna Cortes, and CJ Burges. 2010. MNIST handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist* 2 (2010).

[20] Lucid. 2018. BuildingOS. (2018). https://lucidconnects.com

[21] Johanna L Mathieu, Phillip N Price, Sila Kiliccote, and Mary Ann Piette. 2011. Quantifying changes in building electricity use, with application to demand response. *IEEE Transactions on Smart Grid* 2, 3 (2011), 507–518.

[22] Wes McKinney. 2011. pandas: a Foundational Python Library for Data Analysis and Statistics. (2011).

[23] Clayton Miller and Forrest Meggers. 2017. The Building Data Genome Project: An open, public data set from non-residential building electrical meters. *Energy Procedia* 122 (2017), 439 – 444. https://doi.org/10.1016/j.egypro.2017.07.400 {CISBAT} 2017 International ConferenceFuture Buildings & Districts âĂŞ Energy Efficiency from Nano to Urban Scale.

[24] Natalie Mims, Steven R Schiller, Elizabeth Stuart, Lisa Schwartz, Chris Kramer, and Richard Faesy. 2017. Evaluation of U.S. Building Energy Benchmarking and Transparency Programs: Attributes, Impacts, and Best Practices. (2017). https://doi.org/10.2172/1393621

[25] Andrea Monacchi, Dominik Egarter, Wilfried Elmenreich, Salvatore D'Alessandro, and Andrea M Tonello. 2014. GREEND: an energy consumption dataset of households in Italy and Austria. In *Smart Grid Communications (SmartGridComm), 2014 IEEE International Conference on.* IEEE, 511–516.

[26] David Murray, Lina Stankovic, and Vladimir Stankovic. 2017. An electrical load measurements dataset of United Kingdom households from a two-year longitudinal study. *Scientific data* 4 (2017), 160122.

[27] Pecan St . 2018. Dataport website. (2018). https://dataport.cloud/

[28] Paul Raftery, Shuyang Li, Baihong Jin, Min Ting, Gwelen Paliaga, and Hwakong Cheng. 2018. Evaluation of a cost-responsive supply air temperature reset strategy in an office building. *Energy and Buildings* 158 (2018), 356–370. https://doi.org/10.1016/j.enbuild.2017.10.017

[29] Analytic Rules, Comprehensive Library, Analytic Functions, and Full Programmability. 2012. SkySpark ® Analytic Rules : Combining a Comprehensive Library of Analytic Functions with Full Programmability The Tools Your Need to Address Your Applications. (2012).

[30] Transaction Processing Performance Council. 2018. TPC-C Benchmark Revision 5.11.0. (2018). http://www.tpc.org/tpc_documents_current_versions/current_specifications.asp

[31] Thomas Weng, Anthony Nwokafor, and Yuvraj Agarwal. 2013. BuildingDepot 2.0. *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings - BuildSys'13* (2013), 1–8. https://doi.org/10.1145/2528282.2528285

[32] Tom White. 2012. *Hadoop: The definitive guide.* " O'Reilly Media, Inc.".

[33] S E Womble, J R Girman, E L Ronca, R Axelrad, H S Brightman, and J F Mccarthy. 1995. Developing Baseline Information on Buildings and Indoor Air Quality (BASE '94). Part II (1995), 1–8.