Energy 336 (2025) 138329

Contents lists available at ScienceDirect

Energy

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

AutoControl: An end-to-end fully automated workflow for control design of building energy systems[☆]

Ziqi Hu [a,b,1], Mingchen Li [a,b,1], Hao Tang [c], Zhe Wang [a,b,*]

[a] Department of Civil and Environmental Engineering, The Hong Kong University of Science and Technology, Hong Kong Special Administrative Region of China
[b] HKUST Shenzhen-Hong Kong Collaborative Innovation Research Institute, Futian, Shenzhen, China
[c] Institute of Smart City and Intelligent Transportation, Southwest Jiaotong University, Chengdu 611756, China

ARTICLE INFO

Keywords:
Automated building control
Large language models
Heating
Ventilation and air conditioning systems
Brick schema
Particle swarm optimization

ABSTRACT

Developing an efficient and effective control system is critical to enhance built environment quality, to reduce building energy consumption and to achieve the carbon neutrality goal. However, this process is expertise demanding, time consuming, and error-prone, particularly for large-scale buildings with complicated building energy systems. To address this challenge, this study proposes AutoControl, an end-to-end fully automated workflow based on Large Language Models (LLMs). AutoControl incorporates two LLM-based agents that utilize Brick models as inputs to automatically design and generate Proportional–Integral (PI) controller codes. The first LLM agent, a semantic interpreter, extracts and translates configuration details from the Brick model into a comprehensive description of the building energy system and related control specifications. The second LLM agent, a control expert agent, generates the PI controller code based on the interpreted configurations. Finally, Particle Swarm Optimization (PSO) is employed to fine-tune the parameters of the PI controller. Experiments were conducted on three test cases with diverse HVAC system configurations using the BOPTest virtual testbed. AutoControl achieved an average Mean Absolute Error (MAE) in temperature of 0.323 °C and an average Root Mean Square Error (RMSE) in temperature of 0.766 °C during a week-period, demonstrating robust temperature control performance and strong generalization capabilities. These results highlight the potential of using LLMs for automatic development of building controllers.

1. Introduction

1.1. Building energy control

The control of building energy systems plays a significant role in ensuring indoor environment quality and enhancing building energy efficiency. Existing studies found buildings account for approximately 40% of total energy consumption and over 70% of electricity consumption in the United States, with similar trends observed globally [1,2]. In regard to the building consumption, around 30% of the energy is consumed by heating, ventilation, and air-conditioning (HVAC) systems [3]. Efficient operation and control of these systems is critical to reducing energy waste and mitigating environmental impacts. It has been shown that equipment malfunctions and inefficient control strategies in HVAC and lighting systems can lead to energy waste ranging from 4% to 20% [3], with performance degradation in commercial buildings reaching as high as 15% to 30% [4]. In Addition, inefficient control strategies also have a negative impact on the comfort of building occupants. With around 90% of individuals spending the majority of their time indoors [5], the quality of indoor environment — including thermal comfort, air purity, and lighting, etc. — directly influences their well-being, motivation, performance and contentment [6]. The evaluation of the quality of indoor environment, particularly thermal comfort, is formally defined by industry-established standards such as American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE) Standard 55 [7,8]. Such thermal comfort constraints serve as the baseline for the controller design implemented in AutoControl.

☆ This article is part of a Special issue entitled: 'ICDIES2025' published in Energy.
* Corresponding author at: Department of Civil and Environmental Engineering, The Hong Kong University of Science and Technology, Hong Kong Special Administrative Region of China.
E-mail address: cezhewang@ust.hk (Z. Wang).
[1] Z. Hu and M. Li contributed equally to this work.

https://doi.org/10.1016/j.energy.2025.138329
Received 14 June 2025; Received in revised form 26 August 2025; Accepted 2 September 2025
Available online 16 September 2025

1.1.1. *Rule based control*

Rule based control (RBC) is the most widely used control strategy used in buildings due to its simplicity and robustness [9]. RBC, in particular, relies on predefined heuristic rules to operate building energy systems, making it a default choice in conventional building management systems. Existing studies highlight its application in standardizing control sequences, such as those outlined in ASHRAE Guideline 36 (G36), which provides uniform operational protocols for HVAC systems to enhance energy efficiency [10]. Faulkner et al. [11] demonstrated the implementation of RBC sequences based on G36 within a variable air volume (VAV) system, achieving significant energy efficiency through interactions between air-side and water-side controls. These sequences, rooted in trim-and-respond logic, adjust setpoints based on real-time measurements like damper positions and zone temperatures, illustrating the capacity of RBC to balance energy efficiency and occupant comfort. However, the static rule sets of RBC often fail to adapt to dynamic building conditions or complex system behaviors, leading to suboptimal performance under varying occupancy or climate scenarios [12].

1.1.2. *Model predictive control*

To address the limitation of RBC, model predictive control (MPC) has been developed by leveraging predictive models, predicted disturbances, and real time optimization [13]. MPC employs a receding-horizon strategy, forecasting system states over a control horizon, then determining optimal control actions iteratively to minimize objectives while meeting operational constraints. By rescheduling control sequences at each iterative step, the accumulation of errors can be profoundly reduced [14]. Numerous reports in the literature demonstrated the superior performance of MPC addressing the nonlinear dynamics and multiple objectives inherent in modern HVAC systems [15]. Serale et al. [16] implemented MPC in building thermal control, achieving 15% to 20% average energy savings across diverse building types, alongside significant peak load reductions of approximately 30% when integrated with demand response (DR) strategies. Joe et al. [17] utilized MPC to leverage the thermal mass of buildings as virtual storage, optimizing the operation of residential air conditioning systems. The results demonstrated that, compared to traditional control methods, MPC can save approximately 10.6% in operating costs and achieve a 25% to 46% reduction in peak demand while maintaining indoor comfort.

1.1.3. *Reinforcement learning control*

The implementation of MPC relies on models that can accurately reflect the building thermal dynamics. However, developing and identifying an accurate mode is expertise demanding and time consuming, which limits its adoption in real buildings [18,19]. Reinforcement learning (RL) was proposed to address this challenge. RL learns control policies from real-time interactions with the environment, mitigating the need for predefined models. The HVAC operation in RL is formulated as a Markov decision process (MDP), where an agent iteratively adjusts actions—such as air flow rates in VAV systems—to maximize a cumulative reward, for example, balancing energy cost and occupant comfort [20]. Specifically, integrated with deep learning significantly enhances the capacity of RL to manage nonlinear and uncertain systems, rendering it well-suited for the control of HVAC systems in buildings [21]. Nguyen et al. [22] applied a phasic policy gradient (PPG) algorithm to optimize indoor temperature and energy efficiency under dynamic and unpredictable conditions such as varying weather, occupancy patterns, and equipment performance, achieving up to 52% improvement in temperature comfort metrics, 14% reduction in power consumption, and faster convergence to optimal policies compared to MPC and RBC methods.

Despite RL demonstrates promising ability in handling complex and uncertain HVAC systems, several limitations set barriers for its adoption in automating control design. Specifically, extensive interaction data are required for RL model training, where the data can be difficult to obtain in real-world building due to time and safety constraints. Additionally, training of RL models is sensitive to hyper-parameters and sometimes result in suboptimal or unsafe control sequences. Other advanced approaches, such as expert systems, also face challenges in improving building energy flexibility, due to the reliance on simple manual knowledge engineering [23]. These limitations highlight the demand for an automated, end-to-end and generalizable approach to control design.

1.2. *Applications of LLMs in building industry*

With the rapid developing of artificial intelligence (AI), generative AI, particularly the large language model (LLM), has demonstrated its potential in various fields, such as computer vision [24], speech processing [25], code generation [26], fault diagnosis [27], etc. Specifically, the Generative Pre-trained Transformer (GPT) represents a subset of large language models (LLMs) that leverages the transformer architecture, which is engineered to address a broad range of natural language processing (NLP) tasks. GPTs can generate context-aware, logically coherent text outputs along with a degree of expert knowledge, only requiring natural language prompt [28]. With the emerging of models like GPT-4o [29], LLaMA [30], DeepSeek [31,32], Claude, etc., numerous studies have applied LLMs in the building industry. The potential applications of LLMs in buildings include energy modeling, load prediction, control optimization, etc.

1.2.1. *Building energy modeling*

Establishing building energy models (BEMs) plays a pivotal role in building energy management. By simulating energy dynamics and zone environments based on BEMs, various measures such as sustainability, de-carbonization can be implemented in buildings [33]. However, creating accurate BEMs requires significant expertise including building science and simulation software knowledge (e.g., EnergyPlus [34], etc.), especially for complex and large buildings. Additionally, such a modeling process is time-consuming with parameter tuning and model optimization, which is highly tedious and repetitive. Considering simplifying such processes, several studies have applied LLMs to automatically generate BEMs. Jiang et al. [35] proposed a LLM-based auto-building modeling platform Eplus-LLM, which converts natural language descriptions into EnergyPlus Input Data Files (IDFs) based on a fine-tuned Flan-T5 model (i.e., text-to-text transfer transformers) [36]. The proposed platform achieves rapid IDF generation in approximately 1 min, integrating seamlessly with EnergyPlus for simulation. Model validation demonstrates that Eplus-LLM achieves 100% accuracy, reduces modeling time by over 95% compared to traditional manual approaches, and exhibits robustness across diverse prompt types.

1.2.2. *Building load forecasting*

As a critical component in numerous applications and methodologies, building load forecasting (BLF) has received significant attention in the building field. Particularly when implemented in MPC, the accuracy of BLF directly influences the control performance [37]. With the advancement of artificial intelligence technologies, data-driven BLF have attracted increasing attention, due to the superior performance in building scenarios with incomplete or fragmented information [38,39]. Zhang et al. [40] proposed an automated LLM-based method for building load prediction, which leverages prompting functions to generate Python codes for training and deployment of BLF models. To enhance the performance, the proposed method integrated Bayesian optimization for prompt tuning, and external knowledge bases to provide domain-specific guidance. Additionally, a self-correction mechanism was implemented to rectify coding errors, improving the robustness and accuracy of the prediction. However, the proposed method above was limited to one-step-ahead predictions and struggled with semantic errors.

1.2.3. Building energy control

Effective building energy control strategies, by managing systems such as HVAC, can accelerate the decarbonization process of buildings while ensuring occupancy well-being [41]. For instance, dynamically adjusting HVAC operations based on real-time occupancy and weather conditions enables energy savings and emission reductions while maintaining thermal comfort [42]. With advancements in artificial intelligence, an increasing number of intelligent control systems based on deep learning have emerged. These systems exhibit strong capabilities in nonlinear fitting and dynamic control under uncertainties in complex building environments [43,44]. However, specialized design and training for different buildings results in high technical barriers and implementation costs. The advent of LLMs offers opportunity to streamline this process. An LLM-based agent can interpret varied building data—including weather conditions, indoor environments, occupant behaviors etc.—through natural language inputs, subsequently making decisions or providing recommendations based on expertise knowledge bases. Furthermore, such strategy can adapt to diverse building configurations by leveraging expert knowledge embedded in foundational models [45]. Guo et al. [46] proposed a novel paradigm ControlAgent, which leverages LLM-based agents to automatically design a Proportional–Integral–Derivative (PID) controller for a well-formulated linear time-invariant (LTI) systems. Given the linear ordinary differential equation (ODE) of the system, step-by-step analyses were conducted through natural language, including pole placement, bandwidth design, etc. However, the ControlAgent required accurate system modeling, making it difficult to apply to diverse real-world building scenarios where detailed system parameters or operational data may be incomplete or unavailable.

1.3. Semantic modeling frameworks

Project Haystack provides a tag-centric approach for building operational data; its lightweight tagging conventions make adoption easy across BAS and analytics tools [47]. *The OPC UA Asset Administration Shell (AAS)* is an Industry 4.0 digital-twin metamodel that organizes asset information via sub-models and is commonly realized through an OPC UA companion specification [48,49]. *Brick* is an RDF/OWL ontology for buildings with explicit classes and relationships capturing equipment, points (i.e., sensor/actuator data points), and system topology, enabling semantic consistency and SPARQL-based queries across sites [50]. Our pipeline is knowledge-graph and query-driven: Brick's native RDF/OWL representation and established query patterns (SPARQL) reduce the integration overhead and support reasoning and portability, whereas Haystack's flexible tags and AAS's OPC-UA–first realizations typically require additional mapping to fit semantic-web workflows [51]. Ongoing coordination among Brick, Haystack, and ASHRAE223 further improves cross-framework interoperability while preserving Brick's semantic-web advantages [52,53].

1.4. Virtual testbed of building energy systems

With the growing scale and complexity of building energy systems, virtual testbed has become essential for evaluating proposed control methods. TEASER [54] is an open framework for urban energy modeling and urban-scale analyses of retrofit scenarios, especially concentrating on building model generation and energy demand simulation. It automates the generation of reduced-order models (ROMs) from limited data inputs integrating with standards like CityGML [55] for urban modeling. TwinERGY [56] is a digital twin (DT) platform that creates high-fidelity DTs through dynamic simulation models and real-time data calibration. It leverages cross-leveled IoT data flows to achieve precise energy behavior prediction and analysis. The Hardware-in-the-Loop (HIL) simulation framework [57] is a flexible, modular system that helps build custom simulators to test embedded control systems (ECS), which is achieved by combining real hardware controllers with virtual models of physical processes. BOPTEST [58] is a virtual testbed framework developed for benchmarking and evaluating advanced building control algorithms through standardized test cases. It utilizes a containerized environment to enable rapid algorithm development, and simulation without hardware dependencies. Compared with TEASER, TwinERGY and HIL framework, BOPTEST focuses on advanced control development under different HVAC configurations, which can effectively evaluate the control performance of controllers of energy systems.

1.5. Objectives and contributions

Designing controllers for different buildings is expertise demanding and time consuming, particularly because significant inconsistencies exist in system configurations between buildings. In such cases, extensive customization and parameter tuning become inevitable. Although numerous advanced techniques of artificial intelligence emerged in building industry significantly improve the performance of building control, the black-box mechanism of deep learning and expertise in control principles present a high technical barrier for controller design. Additionally, separately development of each step during building control, for instance, system modeling, controller design, parameter tuning, testing etc., makes controller deployment inefficient and tedious. The recent advancement in LLM provides a unique opportunity to address this challenge.

This study proposes an end-to-end fully automated workflow AutoControl for building controller development, by leveraging the building semantic model and LLMs. A multi-agent framework is introduced in AutoControl: a semantic agent interprets the Brick model which represents the semantic information of the target building; and an expert agent designs the PI controller based on user requirements and building configurations. The parameters of the PI controller are fine-tuned by the particle swarm optimization (PSO) algorithm in an iterative process. AutoControl is a plug-and-play solution, which can be smoothly integrated into the target building. Users input Brick model and necessary requirements in natural language to AutoControl, and then a tailored control solution is automated generated.

The input requirements and semantic-based controller design make AutoControl a promising approach for real-world deployment and controller generalization. By leveraging standardized Brick models and simple user inputs (e.g., temperature setpoint), AutoControl demonstrates a plug-and-play solution, significantly reducing the expertise barrier and time cost compared to traditional controller design which requires refined customization effort. Furthermore, by combining semantic information and domain expertise, AutoControl can easily adopt diverse building HVAC configurations without case-specific algorithmic redesign. The key contributions of this study are summarized as follows:

- An end-to-end fully automated workflow AutoControl is proposed. By leveraging the building semantic information, Large Language Models, and embedded expertise, tailored controllers are automatically generated to meet user requirements.
- An LLM-based semantic agent is developed to interpret the Brick model of the building, by translating the configurations into detailed descriptions of the control requirements.
- Three HVAC control cases are conducted to validate the robustness and generalization of AutoControl, demonstrating superior performance in maintaining thermal comfort.

2. Methodology

2.1. Overview of the workflow

The workflow of AutoControl is shown in Fig. 1. Specifically, in order to make AutoControl capable of plug-and-play, the Brick model of the target building is taken as the inputs of this process. The

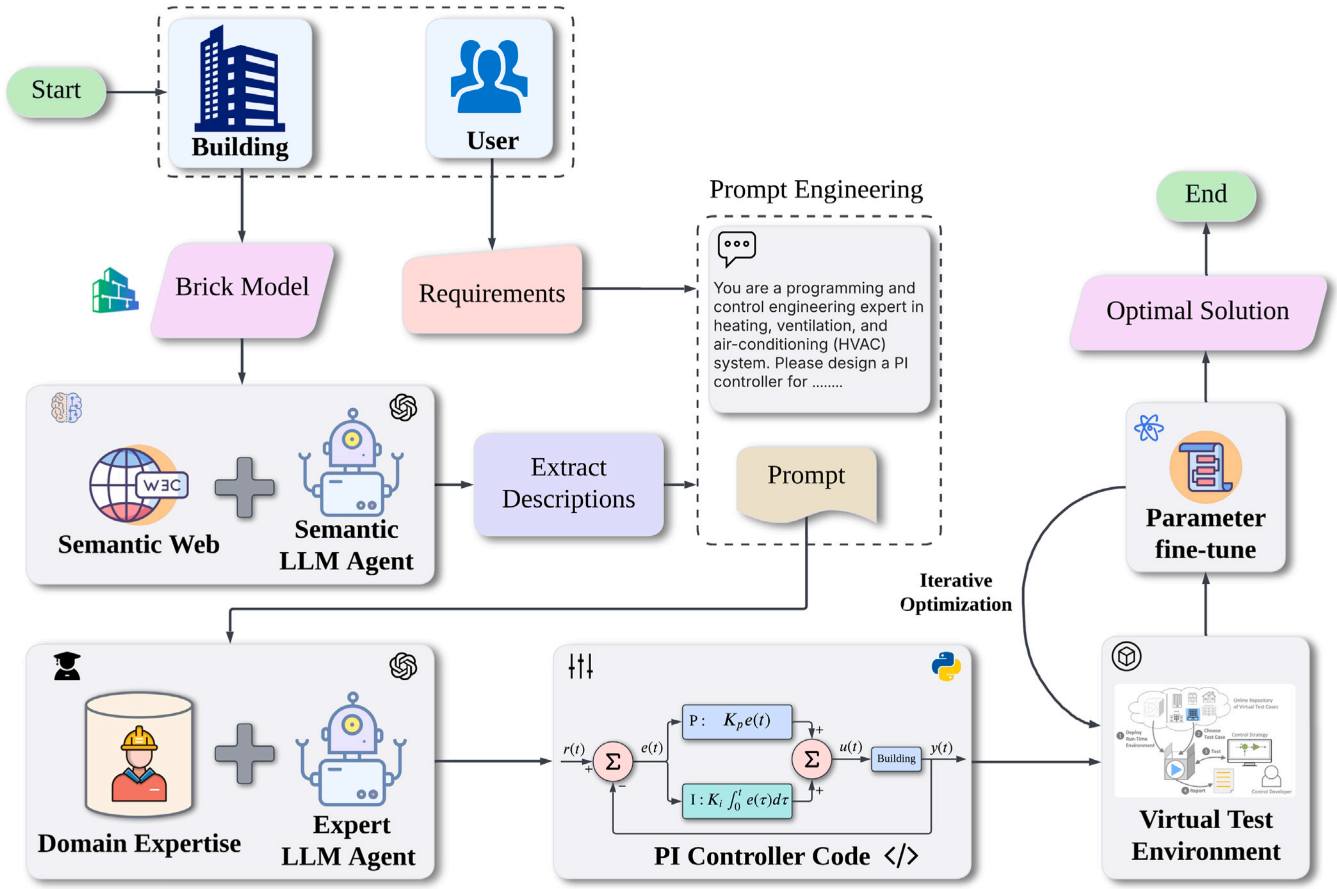


Fig. 1. The overview of the workflow of AutoControl.

Brick model is an open-source framework that standardizes semantic descriptions of building assets and their relationships through an extensible dictionary, relationship definitions, and a flexible data model for seamless integration with existing tools and databases [59]. Using this standardized data format, information about the building energy system can be effectively and comprehensively described.

The formatted key information, such as measurement and control variables, is extracted from the Brick model through the first LLM agent, the semantic agent. Then, the extracted information is integrated into the prompt combined with user requirements. The requirements include controller type, set point temperature, etc. According to the prompt, the Python code of the controller is generated by the second LLM agent, a control expert agent, based on the domain expertise embedded in the official pre-trained model. The initial guesses of the parameters of the controller is also delivered through the second agent. Finally, the controller with initial parameters is tested in the virtual testbed. If the controller with the initial parameters results in unsatisfactory performance, the PSO algorithm will be used for the parameter fine-tuning. Once the control result converges or the number of iterations reaches the maximum number, the system will output the final controller codes.

2.2. Semantic agent

The Semantic Agent utilizes semantic web technologies and large language models (LLMs) to automatically generate comprehensive descriptions of building energy systems, including both control variables and measurement variables. The methodology comprises three sequential steps, depicted in Fig. 2.

Step 1: Variable extraction

In the first stage, the agent identifies and classifies relevant variables from the semantic knowledge graph. Variables are extracted based on their superclass types using subclass reasoning. Specifically, any entity whose type is a subclass of `brick:Sensor` is categorized as a measurement variable and assigned to the "output" category. Conversely, entities of types that are subclasses of `brick:Setpoint` or `brick:Command` are designated as control variables and assigned to the "input" category.

To extract this information, the agent employs the SPARQL query shown in Fig. 3. This query retrieves not only the entity type and classification but also their associated attributes when available.

This query enables a unified extraction of variable types and their associated contextual information, laying the foundation for the subsequent interpretation of attributes and description generation.

Step 2: Property extraction

In the second step, the Semantic Agent parses and extracts key semantic attributes for each identified variable. These include:

- Maximum boundary value (`ref:max`)
- Minimum boundary value (`ref:min`)
- Measurement unit (`ref:unit`)
- Activation condition (`ref:activate`, if applicable)

These attributes are found through external references linked via `ref:hasExternalReference`. This modular referencing allows for consistent reuse and external definition of semantic properties across various entities. The agent consolidates this information to construct a rich semantic profile for each variable, which is subsequently used in the prompt generation for the LLM-based description.

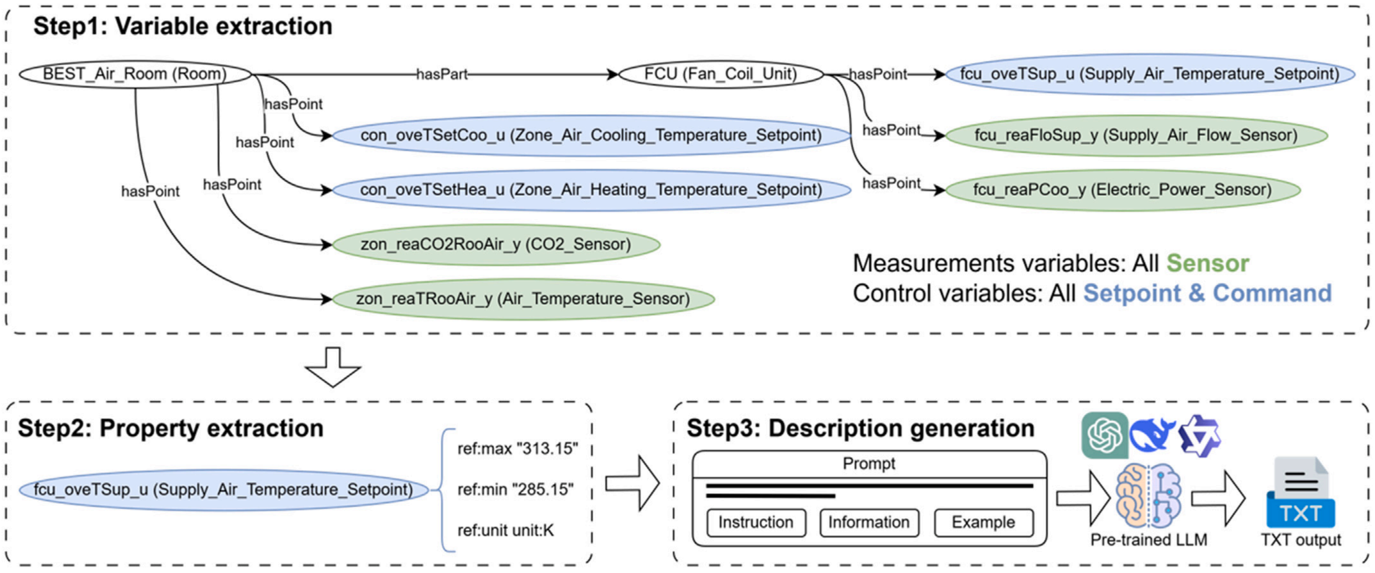

Step1: Variable extraction
BEST_Air_Room (Room)
hasPart
FCU (Fan_Coil_Unit)
hasPoint
fcu_oveTSup_u (Supply_Air_Temperature_Setpoint)
fcu_reaFloSup_y (Supply_Air_Flow_Sensor)
fcu_reaPCoo_y (Electric_Power_Sensor)
con_oveTSetCoo_u (Zone_Air_Cooling_Temperature_Setpoint)
con_oveTSetHea_u (Zone_Air_Heating_Temperature_Setpoint)
zon_reaCO2RooAir_y (CO2_Sensor)
zon_reaTRooAir_y (Air_Temperature_Sensor)
Measurements variables: All Sensor
Control variables: All Setpoint & Command
Step2: Property extraction
fcu_oveTSup_u (Supply_Air_Temperature_Setpoint)
ref:max "313.15"
ref:min "285.15"
ref:unit unit:K
Step3: Description generation
Prompt
Instruction
Information
Example
Pre-trained LLM
TXT
TXT output

Fig. 2. The overview of the semantic LLM-based agent.

```
SELECT ?entity ?type ?maxVal ?minVal ?extRefUnit ?activate ?category
WHERE {
    {
        ?entity rdf:type ?type .
        ?type rdfs:subClassOf* brick:Sensor .
        BIND("output" AS ?category)
    } UNION {
        ?entity rdf:type ?type .
        ?type rdfs:subClassOf* brick:Setpoint .
        BIND("input" AS ?category)
    } UNION {
        ?entity rdf:type ?type .
        ?type rdfs:subClassOf* brick:Command .
        BIND("input" AS ?category)
    }

    OPTIONAL {
        ?entity ref:hasExternalReference ?extRef .
        OPTIONAL { ?extRef ref:max ?maxVal }
        OPTIONAL { ?extRef ref:min ?minVal }
        OPTIONAL { ?extRef ref:unit ?extRefUnit }
        OPTIONAL { ?extRef ref:activate ?activate }
    }
}
```

Fig. 3. The SPARQL query for extracting variables and semantic attributes.

Step 3: Variable description generation

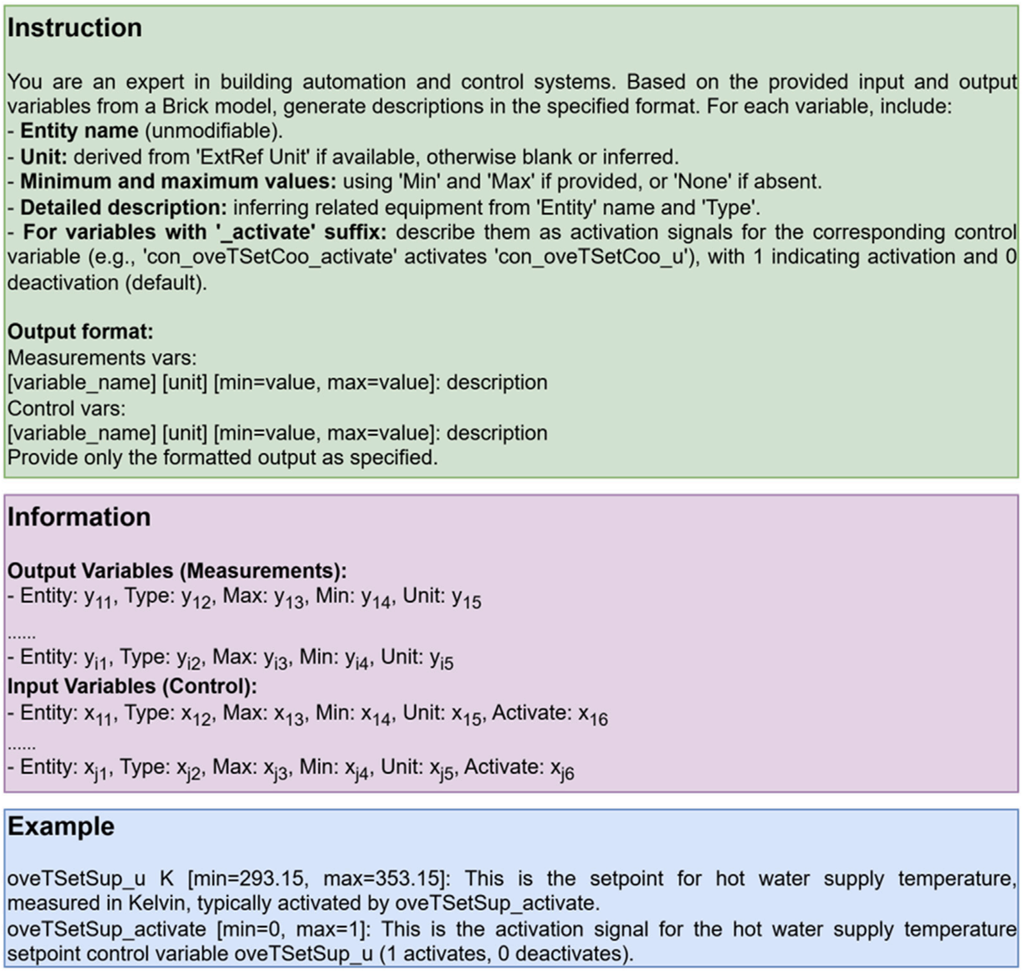
The final stage employs prompt Engineering to leverage the semantic information previously extracted to generate human-readable descriptions of measurement and control variables. The process involves constructing structured prompts comprising clear instructions, detailed semantic information (e.g., variable type, units, boundaries), and illustrative examples. This prompt is then inputted into a large language model, which returns coherent and contextually precise variable descriptions. Fig. 4 illustrates an example of the prompt structure and content fed into the large language model.

The prompt consists of clear instructions, semantic details, and an illustrative example to guide the LLM. To clarify symbols and terms used within the prompt, Table 1 provides explicit definitions.

At the conclusion of this step, the LLM outputs structured and detailed variable descriptions as exemplified in Fig. 5. This representative output demonstrates how the semantic agent generates precise, human-readable definitions for both measurement and control variables, clearly capturing their functionalities and operational boundaries.

Table 1
Symbolic representations of Setpoint/Command and sensor attributes.

Symbol	Description
x_{j1}	Entity name of the jth Setpoint/Command
x_{j2}	Type of the jth Setpoint/Command
x_{j3}	Max value of the jth Setpoint/Command
x_{j4}	Min value of the jth Setpoint/Command
x_{j5}	Unit of the jth Setpoint/Command
x_{j6}	Activate of the jth Setpoint/Command
y_{i1}	Entity name of the ith sensor
y_{i2}	Type of the ith sensor
y_{i3}	Max value of the ith sensor
y_{i4}	Min value of the ith sensor
y_{i5}	Unit of the ith sensor



Instruction

You are an expert in building automation and control systems. Based on the provided input and output variables from a Brick model, generate descriptions in the specified format. For each variable, include:
- **Entity name** (unmodifiable).
- **Unit:** derived from 'ExtRef Unit' if available, otherwise blank or inferred.
- **Minimum and maximum values:** using 'Min' and 'Max' if provided, or 'None' if absent.
- **Detailed description:** inferring related equipment from 'Entity' name and 'Type'.
- **For variables with '_activate' suffix:** describe them as activation signals for the corresponding control variable (e.g., 'con_oveTSetCoo_activate' activates 'con_oveTSetCoo_u'), with 1 indicating activation and 0 deactivation (default).

Output format:
Measurements vars:
[variable_name] [unit] [min=value, max=value]: description
Control vars:
[variable_name] [unit] [min=value, max=value]: description
Provide only the formatted output as specified.

Information

Output Variables (Measurements):
- Entity: y_{11}, Type: y_{12}, Max: y_{13}, Min: y_{14}, Unit: y_{15}

......
- Entity: y_{i1}, Type: y_{i2}, Max: y_{i3}, Min: y_{i4}, Unit: y_{i5}

Input Variables (Control):
- Entity: x_{11}, Type: x_{12}, Max: x_{13}, Min: x_{14}, Unit: x_{15}, Activate: x_{16}

......
- Entity: x_{j1}, Type: x_{j2}, Max: x_{j3}, Min: x_{j4}, Unit: x_{j5}, Activate: x_{j6}

Example

oveTSetSup_u K [min=293.15, max=353.15]: This is the setpoint for hot water supply temperature, measured in Kelvin, typically activated by oveTSetSup_activate.
oveTSetSup_activate [min=0, max=1]: This is the activation signal for the hot water supply temperature setpoint control variable oveTSetSup_u (1 activates, 0 deactivates).

Fig. 4. The prompt template for variable description generation.

2.3. PI controller design

This section introduces the automated process of PI controller design for the HVAC system of the building in detail. PI controllers are widely used in most HVAC control loops [60], due to the robust performance and simple structure. As shown in Fig. 6, the PI controller is applied to track a reference signal $r(t)$ (e.g., set temperature, humidity, CO_2 concentration and etc.) by steering the HVAC output $y(t)$. Specifically, in the HVAC control loop, multiple sensors are used to measure the system output $y(t)$, which are then fed back to the PI controller along with the reference signal $r(t)$ by computing the tracking error $e(t) = r(t) - y(t)$. Then, the PI controller computes the control signal $u(t)$ of the HVAC system by integrating the proportional and integral time parameters K_p and K_i with the tracking error $e(t)$. The kth step of control signal $u(k)$ is formulated as:

$$u(k) = K_p e(k) + K_i \sum_{j=0}^{k} e(j), \tag{1}$$

where, $e(j)$ represents the tracking error at time step j. It should be noted that, the PI controller in the HVAC system adopts discrete-time implementation to align with the digital control hardware requirements. In this study, the measurement of the system is limited to indoor temperature, with the PI controller designed specifically for set-point tracking.

The architecture of such a control loop is also known as closed-loop control, where the measurements of the system are fed back to the controller to calculate the input of the system. According to experience in modern control engineering, the PI controller can be designed based on the transfer function of the dynamic system, which is formulated as:

$$G(s) = \frac{b_m s^m + b_{m-1} s^{m-1} + \cdots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0}, \tag{2}$$

where, s represents the complex frequency variable in the Laplace domain, a and b are the coefficients of output and input signals, respectively. In the real-world physical system, the $m \leq n$ is usually adopted as a common assumption. As for HVAC systems, the 2R1C model [61] is widely adopted as the thermal model of the zone, which can be formulated as:

$$G(s) = \frac{T_{\text{out}}(s)}{T_{\text{in}}(s)} = \frac{1}{1 + \frac{R_{\text{in}}}{R_{\text{out}}} + s R_{\text{in}} C_{\text{out}}}, \tag{3}$$

where, $T_{\text{out}}(s)$ and $T_{\text{in}}(s)$ are the outdoor and indoor temperature in the Laplace domain, respectively. R_{in} and R_{out} are the thermal resistance, and C_{out} is the thermal capacity. The structure of the 2R1C model is illustrated in Fig. 7. Based on the transfer function of the system, the PI controller can be well-designed given the requirements of the phase margin and settling time [46]. However, in the real-world system, the thermal model is usually complex and dynamic and it is difficult to obtain accurate values of the parameters in the model, such as thermal resistance R and capacity C.

Therefore, instead of modeling the HVAC system, an end-to-end PI controller design strategy is proposed based on prompt engineering [62]. After obtaining the specific configuration from the first semantic agent, the second LLM-based agent, endowed with control and HVAC expertise, will generate the PI controller in the form of Python code. The prompt fed to the second agent includes five sections.

```
Measurement variables:
- reaQHea_y [W, min=None, max=None]: Heating thermal power sensed value, measured in watts.
- reaPPum_y [W, min=None, max=None]: Electric power sensed value for pump, measured in watts.
- reaCO2RooAir_y [PPM, min=None, max=None]: CO2 concentration sensed in room air, measured in parts
    per million.
- reaTRoo_y [K, min=None, max=None]: Air temperature sensed in the room, measured in Kelvin.

Control variables:
- oveTSetSup_u [K, min=293.15, max=353.15]: Setpoint for hot water supply temperature, measured in
    Kelvin, activated by oveTSetSup_activate.
- oveTSetSup_activate [min=0, max=1]: Activation signal for oveTSetSup_u (1 activates, 0 deactivates).
- oveTSetCoo_u [K, min=278.15, max=308.15]: Setpoint for zone air cooling temperature, measured in
    Kelvin, activated by oveTSetCoo_activate.
- oveTSetCoo_activate [min=0, max=1]: Activation signal for oveTSetCoo_u (1 activates, 0 deactivates).
- oveTSetHea_u [K, min=278.15, max=308.15]: Setpoint for zone air heating temperature, measured in
    Kelvin, activated by oveTSetHea_activate.
- oveTSetHea_activate [min=0, max=1]: Activation signal for oveTSetHea_u (1 activates, 0 deactivates).
- ovePum_u [min=0, max=1]: On/off command for pump (1 turns on, 0 turns off), activated by
    ovePum_activate.
- ovePum_activate [min=0, max=1]: Activation signal for ovePum_u (1 activates, 0 deactivates).
```

Fig. 5. The example of generated variable descriptions from the semantic LLM-based agent.

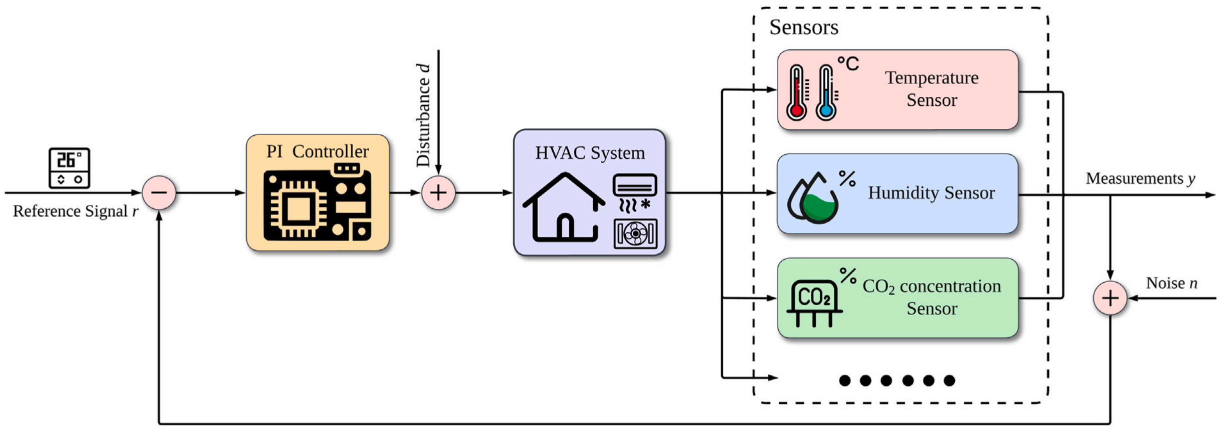


Fig. 6. Feedback control of the PI controller in the HVAC system.

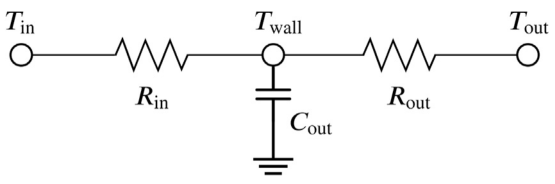


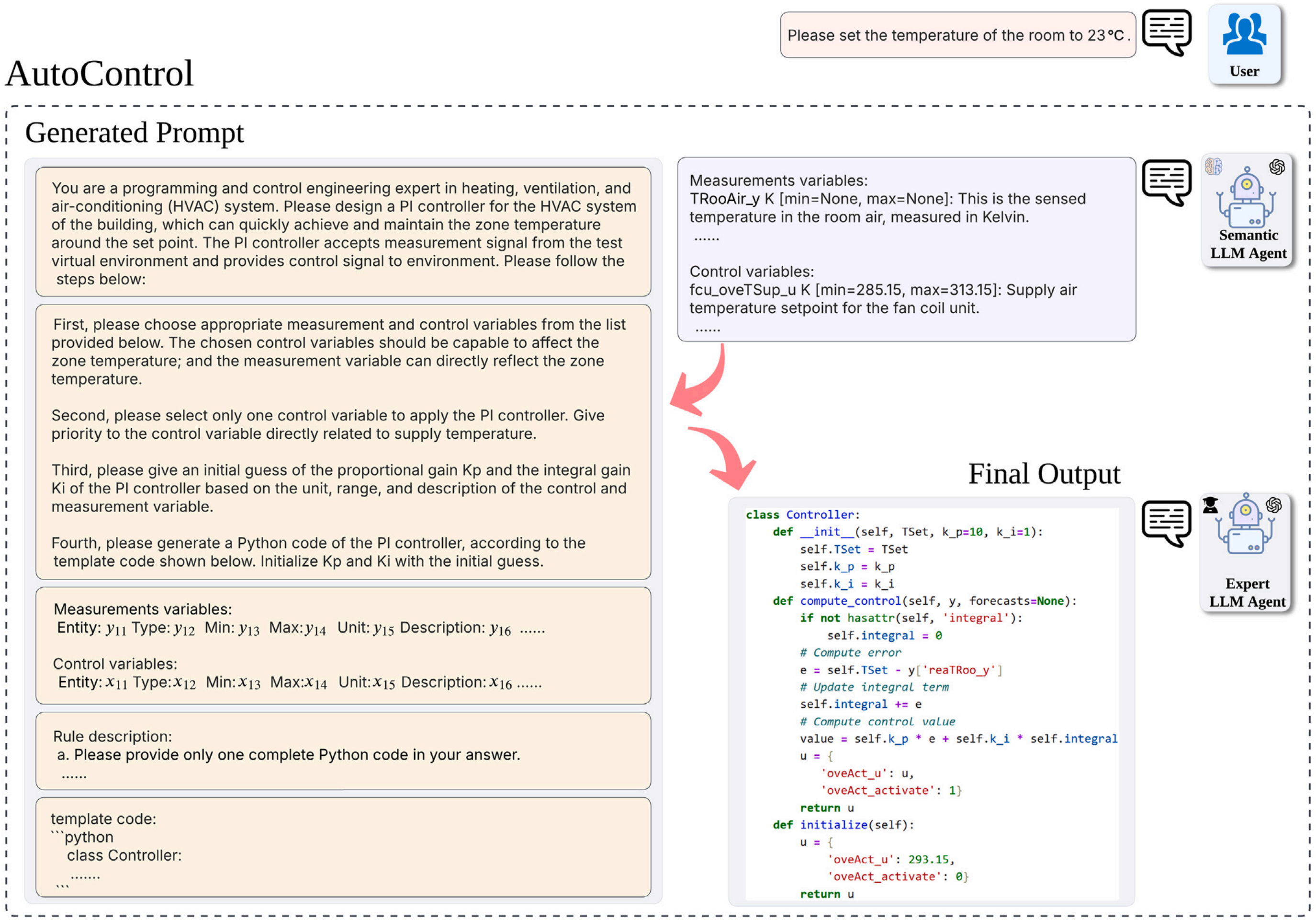
Fig. 7. The structure of the 2R1C thermal model.

1. The first section is the task description, outlining the primary objective of designing a PI controller for an HVAC system. In this study, the goal of control is to achieve and maintain the zone temperature around a predefined setpoint.
2. The second section is a step-by-step instruction, providing a structured approach to developing the PI controller.
3. The third section enumerates the available measurement and control variables extracted from the first semantic agent, detailing their units, ranges, and descriptions.
4. The fourth section is the rule description, specifying the format for the output of the agent, and some specific requirements.
5. The fifth section is the template code, providing a Python class structure for implementing the PI controller.

For illustration, Fig. 8 shows an example of the prompt provided from the user.

As for the second section of the prompt, it includes four key steps: (1) selecting and verifying the appropriate measurement and control variables from a given list (described in the third section), (2) choosing a single control variable for the PI controller, prioritizing those related to the supply temperature, (4) providing an initial estimation for K_p and K_i based on the unit, range, and description of the selected variables, and (5) generating a Python controller class using the template provided. Note that, in this study, the single input system is adopted, which has only one control variable. The rest of the control variables are set as constant. To further simplify the problem, the selection of control variables is constrained to temperature-related variables, such as the supply air temperature for the Fan Coil Unit (FCU). This is achieved by incorporating relevant keywords into the prompts provided to the LLM agent.

In the third section of the prompt, the outputs of the first semantic LLM agent are inserted into the prompt. The elaborated information includes name, unit, range and description about the measurement and control variables provides comprehensive understanding of the



Please set the temperature of the room to 23 °C .

You are a programming and control engineering expert in heating, ventilation, and air-conditioning (HVAC) system. Please design a PI controller for the HVAC system of the building, which can quickly achieve and maintain the zone temperature around the set point. The PI controller accepts measurement signal from the test virtual environment and provides control signal to environment. Please follow the steps below:

First, please choose appropriate measurement and control variables from the list provided below. The chosen control variables should be capable to affect the zone temperature; and the measurement variable can directly reflect the zone temperature.

Second, please select only one control variable to apply the PI controller. Give priority to the control variable directly related to supply temperature.

Third, please give an initial guess of the proportional gain Kp and the integral gain Ki of the PI controller based on the unit, range, and description of the control and measurement variable.

Fourth, please generate a Python code of the PI controller, according to the template code shown below. Initialize Kp and Ki with the initial guess.

Measurements variables:
Entity: y_{11} Type: y_{12} Min: y_{13} Max: y_{14} Unit: y_{15} Description: y_{16} ......

Control variables:
Entity: x_{11} Type: x_{12} Min: x_{13} Max: x_{14} Unit: x_{15} Description: x_{16} ......

Rule description:
a. Please provide only one complete Python code in your answer.
......

template code:
```python
class Controller:
    .......
```

Measurements variables:
TRooAir_y K [min=None, max=None]: This is the sensed temperature in the room air, measured in Kelvin.
......

Control variables:
fcu_oveTSup_u K [min=285.15, max=313.15]: Supply air temperature setpoint for the fan coil unit.
......

```python
class Controller:
    def __init__(self, TSet, k_p=10, k_i=1):
        self.TSet = TSet
        self.k_p = k_p
        self.k_i = k_i
    def compute_control(self, y, forecasts=None):
        if not hasattr(self, 'integral'):
            self.integral = 0
        # Compute error
        e = self.TSet - y['reaTRoo_y']
        # Update integral term
        self.integral += e
        # Compute control value
        value = self.k_p * e + self.k_i * self.integral
        u = {
           'oveAct_u': u,
           'oveAct_activate': 1}
        return u
    def initialize(self):
        u = {
            'oveAct_u': 293.15,
            'oveAct_activate': 0}
        return u
```

Fig. 8. The example of the generated prompt and the workflow of the expert LLM-based agent.

control problem for the second LLM-based expert agent. The fourth section of the prompt is based on the empirical knowledge of prompt engineering as shown in Table 2. Rule 1 emphasizes that only the Python code should be returned without any additional explanations. This constraint ensures consistency and adherence to the task requirements, facilitating seamless integration of the generated code into the virtual environment. Rule 2 is required by the virtual environment, where the activation state should be set to 1 before it can be used. Rule 3 is applied to ensure that none of the control variables is null, as these variables are usually assigned a default value of 0 in the virtual environment, which may cause problems in the control process. Rule 4 is provided by the user. Rule 5 and Rule 6 are designed for the multiple-zone case where the PI controller is required to be tailored for each individual zone. The fifth section of the prompt includes methods for initialization and control computation, with placeholders for measurement and control variables. The template serves as a foundational framework, enabling the user to customize and implement the controller efficiently. By following this template, users can ensure that the controller complies with the specified design requirements and integrates seamlessly with other HVAC systems. The example code template is illustrated in Fig. A.1, Appendix A.1.

2.4. Parameter fine-tuning

Fine-tuning of the parameters K_p and K_i is critical for the PI controller design, since the control performance such as settling time, steady-state errors, and etc., relies heavily on the values of the parameters. In traditional control engineering, this process generally requires substantial human expertise and considerable manual intervention. In this study, the particle swarm optimization (PSO) algorithm is applied to alleviate the work load of this process. There are two main reasons that PSO is conducted as the fine-tune algorithm: (1) both in virtual testbed and real-world building, gradient information on the thermal dynamics is difficult to obtain, which makes gradient-based optimization algorithm hard to implement; (2) the population-based strategy of PSO inherently supports parallel simulation evaluations, which can greatly improve optimization efficiency.

The auto-generated PI controller with initial parameters is first evaluated in the virtual testbed. Subsequently, the controller parameters are

Table 2
The rule-based external knowledge for prompt engineering.

No.	Rule descriptions
1	Please provide only one complete Python code in your answer.
2	Set the activation state variables of all control variables to 0 in the initialization stage, and set all the activate state variables to 1 in the computation stage.
3	All of the remained control variables should be set to reasonable fixed values, without being undefined in the output dict.
4	Set the reference temperature of the controller to 22.5 °C.
5	Please identify the number of zones from the configurations of the variables (There may only one zone or multiple zones). If there is one thermal zone, the class name should be Controller. If there are multiple thermal zones, the class name should be Controller_Zone_$key_word_in_variables$.
6	Design the PI controller separately for each zone in separate codes. Return all the PI controller codes for the zones identified.

iteratively optimized using the PSO algorithm, with performance re-evaluated in the testbed after each optimization cycle. Specifically, the PSO begins by initializing a population of particles, where the position of each particle represents a potential solution in the form of a $[K_p, K_i]$ pair, where K_p and K_i are the proportional and integral gain of the PI controller, respectively. These particles are randomly distributed within predefined bounds for K_p and K_i. During each iteration, the performance of each particle is evaluated using an objective function that quantifies the control system's performance. The objective function is given by

$$f = w_1 \cdot e_{\text{avg\_error}} + w_2 \cdot e_{\text{over\_shoot}} + w_3 \cdot e_{\text{settling\_time}} \tag{4}$$

where, $e_{\text{avg\_error}}, e_{\text{over\_shoot}}, e_{\text{settling\_time}}$ are the average temperature deviation, over-shoot, and settling-time of the temperature control result, respectively. $[w_1, w_2, w_3]$ are weighting factors. In this study, $[w_1, w_2, w_3]$ is set to $[0.8, 0.1, 0.1]$. The definitions of $e_{\text{avg\_error}}$, $e_{\text{over\_shoot}}$, and $e_{\text{settling\_time}}$ are given in Appendix A.2.

Then, the positions and velocities of particles are updated based on their individual optimal positions $x^p = [K_p^p, K_i^p]$ and the global optimal position $x^g = [K_p^g, K_i^g]$ across the entire population. Specifically, for each particle, the velocity is updated using the following equation:

$$\begin{aligned} v_{i,j}(t+1) = \gamma \cdot v_{i,j}(t) + c_1 \cdot r_1 \cdot (x_{i,j}^p - x_{i,j}(t)) \\ + c_2 \cdot r_2 \cdot (x_j^g - x_{i,j}(t)) \end{aligned} \tag{5}$$

where, $v_{i,j}(t)$ and $x_{i,j}(t)$ are the velocity and position of the ith particle in the jth dimension at iteration t, respectively, γ is the inertia weight, which controls the influence of the previous velocity on the current update, c_1 and c_2 are the cognitive and social learning rate, respectively, r_1 and r_2 are random factors uniformly distributed in the range $[0, 1]$, introducing stochasticity to the search process, γ is the inertia weight, $x_{i,j}^p$ is the personal best position of the ith particle in the jth dimension, and x_j^g is the global best position in the jth dimension across all particles. After updating the velocity, the position of each particle is updated as follows:

$$x_{i,j}(t+1) = x_{i,j}(t) + v_{i,j}(t+1) \tag{6}$$

To enhance the optimization performance, a dynamic hyper-parameter tuning strategy is implemented. Specifically, the inertia weight γ, which controls the balance between exploration and exploitation, is linearly decreased from an initial maximum value $\gamma_{\max} = 0.9$ to a minimum value $\gamma_{\min} = 0.4$ during the iterations. Similarly, the cognitive learning rate c_1 gradually decreases from $c_1^{\max} = 2.5$ to $c_1^{\min} = 1.5$, while the social learning rate c_2 gradually increases $c_2^{\min} = 1.5$ to $c_2^{\max} = 2.5$. This dynamic adjustment implies that the swarm initially emphasizes individual particle exploration and later shifts its focus toward exploiting the global optimal solution. This adaptive mechanism ensures that the algorithm maintains a strong exploration capability in the early stages while gradually shifting toward exploitation as the search progresses. The PSO optimization is summarized in Algorithm 1. In this paper, the particle number N is set to 15, the maximum iteration $N_{\max}$ is set to 10. Furthermore, parallel computing is employed to evaluate the performance of multiple particles simultaneously, significantly reducing the computational time required for the optimization process.

Algorithm 1 PSO-Based Parameter Tuning

```
Require: Particles $N$, Iterations $N_{\max}$, Setpoint $T_{\text{set}}$
Ensure: Optimal $K_p$, $K_i$
 1: Initialize particles: Random positions $\mathbf{x}_i$, velocities $\mathbf{v}_i$
 2: Initialize $x_i^p \leftarrow \mathbf{x}_i$, $x^g \leftarrow \emptyset$, $f_i^p \leftarrow \infty$, $f^g \leftarrow \infty$
 3: for $t = 1$ to $N_{\max}$ do
 4:     Update $\gamma, c_1, c_2 \leftarrow$ get_dynamic_params($t, N_{\max}$)
 5:     for each particle $i$ do
 6:         Evaluate fitness $f_i \leftarrow$ evaluate_controller($\mathbf{x}_i, T_{\text{set}}$)
 7:         if $f_i < f_i^p$ then
 8:             $x_i^p \leftarrow \mathbf{x}_i$, $f_i^p \leftarrow f_i$
 9:         end if
10:         if $f_i < f_i^g$ then
11:             $x^g \leftarrow \mathbf{x}_i$, $f^g \leftarrow f_i$
12:         end if
13:     end for
14:     for each particle $i$ do
15:         $\mathbf{v}_i \leftarrow \gamma\mathbf{v}_i + c_1 r_1 (x_i^p - \mathbf{x}_i) + c_2 r_2 (x^g - \mathbf{x}_i)$
16:         $\mathbf{x}_i \leftarrow$ clamp($\mathbf{x}_i + \mathbf{v}_i$, bounds)
17:     end for
18: end for
19: return $(K_p, K_i) \leftarrow x^g$
```

3. Results

In this section, a comprehensive set of experiments is conducted in the Building Optimization Testing Framework (BOPTEST) [58] to evaluate the performance of AutoControl for temperature control of the HVAC system. BOPTEST is an open-source virtual environment developed to facilitate simulation of building HVAC control strategies. It employs Modelica models [63] to simulate realistic building physical dynamics. To ensure rapid and reproducible execution with diverse control algorithms, it incorporates Docker containerized deployment - a lightweight virtualization technology that packages software with all the dependencies into standardized units. Additionally, extensive sets of test cases are embedded in the platform, with diverse system types, such as single-zone residential buildings with radiant heating or multi-zone setups with varied HVAC configurations. These test cases allow for the overwriting of supervisory and local-loop control signals, and provide predefined testing scenarios, including specific time periods and boundary conditions like weather and energy pricing. Notably, the weather-related data are not utilized in the simulation to evaluate the robustness of AutoControl in unpredictable weather conditions. The mean absolute error (MAE), root mean square error (RMSE) and thermal discomfort (ΔT_d) are used to evaluate the temperature control performance. The definitions of the metrics are shown in Appendix A.3. The control performance of AutoControl is compared with the baseline controller embedded in the BOPTEST and the Model Predictive Control (MPC). The formulation of MPC is given in Appendix A.4.

To evaluate the generalization ability of AutoControl, we select three test cases — *BESTEST_Air*, *BESTEST_Hydronic*, and *Two_Zone_Apartment_Hydronic* — representing different air conditioning terminals (Fan Coil Units or radiators), different control variables (`fcu_oveTSup_u` or `oveTSetSup_u`), and different numbers of thermal zones. The detailed properties of the three test cases have been illustrated in Appendix A.5. Since the official BOPTEST platform does not currently provide semantic Brick models for its benchmark test cases, we first constructed detailed Brick-based semantic models for the three selected BOPTEST cases.

To construct these Brick-based models, we developed a structured and reproducible modeling process using programmatic rules rather than manual annotation. For each test case, we first defined key physical entities such as rooms, fan coil units (FCUs), heating and cooling coils, and lighting or plug loads. These components were then described using standardized Brick classes and connected through well-defined relationships (e.g., a fan coil unit "feeds" a room, or a sensor "measures" a temperature). In addition to physical equipment, we systematically defined the control inputs and sensor outputs relevant to each system. Each point was annotated with detailed metadata, including its engineering unit, valid range, and its link to the simulation variable in BOPTEST. To ensure consistency and enable downstream automation, all of these elements — including equipment, points, and their relationships — were generated using a rule-based Python script and exported in the RDF Turtle format, following Brick schema conventions.

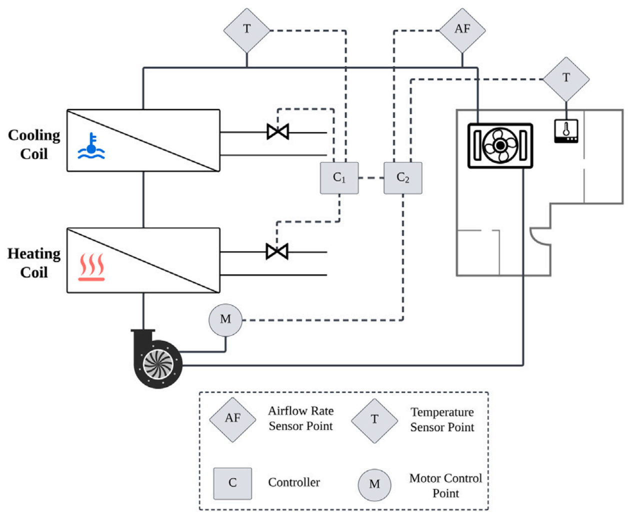


Fig. 9. The overview of the HVAC system for the BESTEST_Air test case.

The resulting semantic representations for these cases are presented in Appendix A.7. Additionally, to ensure semantic consistency and correctness, these three models were validated using the Ontology Validation feature provided by the BuildingMOTIF toolkit [64]. All three Brick models successfully passed ontology validation, confirming their semantic compliance and suitability for subsequent experimentation.

The OpenAI GPT-4o is utilized for the two LLM agents, the semantic agent and the expert agent. The temperature parameter of the expert LLM agent is set to 0.0, since the randomness is unnecessary for the PI controller code generation task.

3.1. Single zone with FCU

The building is a single-room zone with a floor area of 6 m × 8 m, a floor-to-ceiling height of 2.7 m, four exterior walls, a flat roof, and two 3 m × 2 m windows on the south wall, designed for two occupants with a light load density. The climate data is collected near Denver, CO, USA with a latitude of 39.76° and longitude of −104.86°. The zone is equipped with an idealized four-pipe FCU for heating and cooling, which includes a fan, cooling coil, heating coil, and filter. The room air is drawn by the fan, conditioned by passing through the coils and the filter, and then supplied back to the room. The cooling coil uses chilled water from a chiller, while the heating coil uses hot water from a gas boiler, with the fan motor controlled by a variable speed drive. The system schematic diagram is given in Fig. 9. Note that there is an ideal controller embedded in the test case, which can accurately control the supply air temperature through the coils.

An example of generated variable descriptions of the semantic LLM-based agent is illustrated in Fig. A.2, Appendix A.6. As for the temperature control, the tracking temperature point was set to 22.5 °C. The cooling temperature setpoints were 24 °C and 30 °C when the zone was occupied and unoccupied, respectively. The heating temperature setpoints were 21 °C and 15 °C when the zone was occupied and unoccupied, respectively. Fig. 10 illustrates the comparison of the control results between AutoControl with $K_p = 0.01$ and $K_i = 1.94$, the baseline controller and MPC. Note that the temperature was maintained at 22.5 °C for the majority time in a week. Table 3 presents the quantitative comparison between AutoControl, the baseline controller and MPC. AutoControl demonstrated superior performance with an average absolute temperature deviation below 0.2 °C and significant reduction of thermal discomfort compared with the baseline controller and MPC. When conducting MPC, the actuator model of the HVAC system is required to be fitted first. In this test case, the actuator

Table 3
The MAE, RMSE and thermal discomfort of indoor temperature using the baseline controller, MPC and AutoControl in the BESTEST_Air test case.

	Baseline	MPC	AutoControl
MAE [°C]	2.071	4.035	**0.107**
RMSE [°C]	2.536	4.871	**0.770**
ΔT_d [Kh/zone]	3.273	12.161	**0.598**

Table 4
The MAE, RMSE and thermal discomfort of indoor temperature using the baseline controller, MPC and AutoControl in the BESTEST_Hydronic test case.

	Baseline	MPC	AutoControl
MAE [°C]	1.764	0.969	**0.051**
RMSE [°C]	2.127	1.241	**0.335**
ΔT_d [Kh/zone]	2.759	**2.464**	2.609

refers to the fan coil unit. However, the simulation data provided by BOPTEST was insufficient to achieve high-accuracy modeling, where the $R^2 = 0.72$. Therefore, MPC showed large thermal discomfort in this test case.

3.2. Single zone with hydronic radiator

The building is a single-room zone with a floor area of 6 m × 8 m, a floor-to-ceiling height of 2.7 m, four exterior walls, a flat roof, and two south-facing windows of 6 m^2 each, designed for one occupant under a light internal load density. The climate data were collected at Brussels, Belgium. The zone is equipped with a single radiator with thermostatic valve, a circulation pump and a water heater for heating. The nominal thermal power of the radiator and the maximum thermal power of the heater are both 5 kW. The efficiency of the gas heater is determined using a polynomial curve, while a PI controller modulates the supply water temperature within the range of 20 °C to 80 °C to maintain the operative zone temperature. The system schematic diagram is illustrated in Fig. 11.

The example of generated variable descriptions of the semantic LLM-based agent is illustrated in Fig. A.3, Appendix A.6. In this case, the terminal unit differs from the BESTEST_Air test case, featuring a hydronic radiator instead of an FCU. The control variable shifts from the supply air temperature to supply water temperature, which is automatically determined by AutoControl. The temperature setpoints are kept same with the BESTEST_Air test case. Fig. 12 illustrates the comparison of the control results between AutoControl with $K_p = 3.74$ and $K_i = 8.71$, the baseline controller and MPC. The temperature rapidly converges to 22.5 °C after transient oscillations and is maintained at around 22.5 °C in a typical week. Table 4 shows the MAE, RMSE and thermal discomfort of the indoor temperature, where AutoControl achieved smaller MAE and RMSE and similar ΔT_d compared with MPC. The temperature control results of the single zone with FCU and hydronic radiator demonstrate that AutoControl is capable of dealing with different terminal units in buildings.

3.3. Multiple zones with hydronic heat pump

The building is built based on a two-room apartment model, which consists of two rooms with one bathroom. Two thermal zones are considered in this case, which are the night-zone (including the bathroom) and day-zone, respectively. The floor-to-ceiling height of each room is 2.7 m, and the floor areas of the night-zone and day-zone are 22.62 m^2 and 22.02 m^2, respectively. The climate data is collected near Milan, Italy, located at 45.44°N latitude and 9.27°E longitude. The HVAC system consists two floor heating circuits, with on-off valves for the controlling of each thermal zone. The nominal heat capacity of the air-source heat pump is 5 kW. The system schematic diagram is illustrated in Fig. 13.

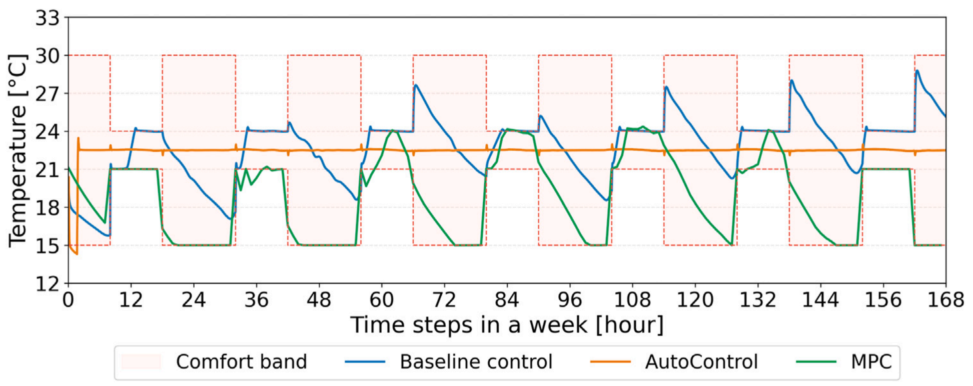


Fig. 10. The comparison of indoor temperature controlled by the baseline controller, MPC and AutoControl during a week for the BESTEST_Air test case.

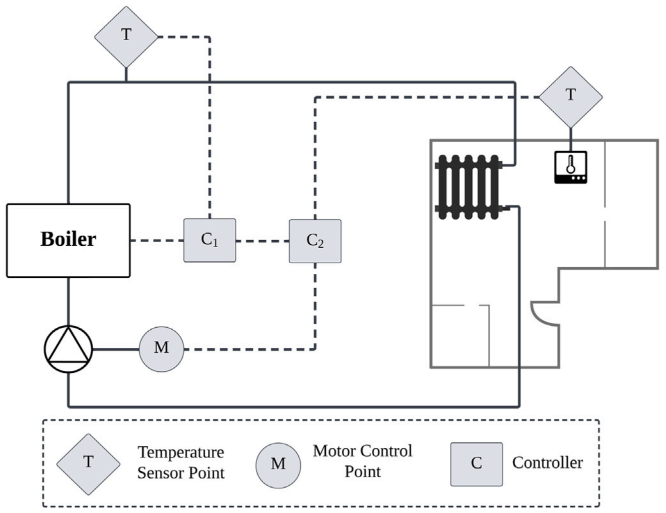


Fig. 11. The overview of the HVAC system for the BESTEST_Hydronic test case.

The temperature control for the multiple-zone is more complex compared to the single-zone. which requires additional system identification steps. The second expert agent needs to identify the number of zones from the outputs of the first semantic agent, where distinct variable prefixes uniquely identify each thermal zone. For example, in this case, the variables *dayZon_reaTRooAir_y* and *nigZon_reaTRooAir_y* represent the measured air temperatures in the day-zone and night-zone, respectively. The second expert agent interprets this nomenclature to determine the number of zones. Accordingly, dedicated PI controllers are implemented for each zone based on these variable definitions. An example of generated variable descriptions of the semantic LLM-based agent is illustrated in Figs. A.4, A.5 and Appendix A.6. The tracking temperature point was set to 23 °C. The cooling temperature setpoints were 25 °C and 30 °C when the zone was occupied and unoccupied, respectively. The heating temperature setpoints were 21 °C and 16 °C when the zone was occupied and unoccupied, respectively. Fig. 14 demonstrates the comparison of the temperature control results of AutoControl with K_p = 2886.08 and K_i = 0.01, the baseline controller and MPC. Note that, despite of localized oscillations, both day-zone and night-zone maintained within the comfort band for the majority of the time in a week. Table 5 demonstrates the quantitative results of AutoControl compared with the baseline controller and MPC. AutoControl$_d$ and AutoControl$_n$ refer to the results of day-zone and night-zone, respectively. The experimental results demonstrate the excellent potential of AutoControl for handling multi-zone HVAC systems.

The impact of Brick scheme variations on parsing accuracy is explored in this test case. The Brick model is modified manually to

Table 5
The MAE, RMSE and thermal discomfort of indoor temperature using the baseline controller, MPC and AutoControl in the Two_Zone_Apartment_Hydronic test case.

	MAE [°C]	RMSE [°C]	ΔT_d [Kh/zone]
Baseline	1.96	2.280	17.2379
MPC	1.007	1.347	19.110
AutoControl$_d$	0.572	1.01	**12.3171**
AutoControl$_n$	**0.562**	**0.95**	

simulate different scenarios in practical implementations. Four scenarios are assumed here, which are Original, no Unit, no Range and no Unit&Range, which represent different absence situations of the related properties of points in the Brick model. Taking the control variable *hydronicSystem_oveMpumCon_u* as an example, the natural language description generated from semantic agent remains consistent under the variations of the Brick model as shown in Table 6, which demonstrates the robustness of the semantic LLM-based agent on Brick model parsing.

The API costs for the first semantic agent and the second expert agent are calculated as shown in Table 7. It costs totally US $0.024 for Case 1 (BESTEST_Air), US $0.019 for Case 2 (BESTEST_Hydronic) and US $0.046 for Case 3 (Two_Zone_Apartment_Hydronic), respectively, using GPT-4o with a unit price of US $2.50/1M input tokens and US $10.00/1M output tokens, accoriding to the pricing of OpenAI in March 2025 (OpenAI API Pricing). Additionally, the computational efficiency of AutoControl is evaluated for three test cases, with the execution time summarized in Table 8. Noted that most of the execution time of AutoControl is spent on the PSO optimization phase across all three test cases. However, it does not indicate that the PSO optimization is inefficient. In fact, the optimization process involves iterative interactions with the virtual testbed, where each evaluation step requires simulation results from BOPTEST. Therefore, the optimization time is heavily influenced by the simulation efficiency of BOPTEST. Nevertheless, the proposed method demonstrates efficient PI controller design, with an execution time under 10 min in multi-threaded configuration and under 60 min in single-threaded configuration. Based on general engineering experience, conventional manual design typically requires 3–5 days to complete the entire process, including system analysis, controller code implementation, and manual fine-tuning by experienced engineers. Generally, the adoption of automated PI controller design can lead to significant labor and time cost savings in building control applications.

4. Discussion

4.1. Contributions

The contributions of the proposed AutoControl are threefold. First, AutoControl is highly automated, with an end-to-end ability of designing the PI controller directly from Brick models of the target building.

Table 6
The generated descriptions of the control variable *hydronicSystem_oveMpumCon_u* under four scenarios of the Brick model.

Scenarios	Descriptions generated from the semantic LLM-base agent
Original	*hydronicSystem_oveMpumCon_u* K [min = 0.0, max = 5.0]: This is the override control setpoint for the hot water flow rate in the pump of the hydronic system, typically activated by *hydronicSystem_oveMpumCon_activate*.
no Unit	*hydronicSystem_oveMpumCon_u* [min = 0.0, max = 5.0]: This is the override control setpoint for the hot water flow rate in the pump of the hydronic system, typically activated by *hydronicSystem_oveMpumCon_activate*.
no Range	*hydronicSystem_oveMpumCon_u* K: This is the override control setpoint for the hot water flow rate in the pump of the hydronic system, typically activated by *hydronicSystem_oveMpumCon_activate*.
no Unit&Range	*hydronicSystem_oveMpumCon_u*: This is the override control setpoint for the hot water flow rate in the pump of the hydronic system, typically activated by *hydronicSystem_oveMpumCon_activate*.

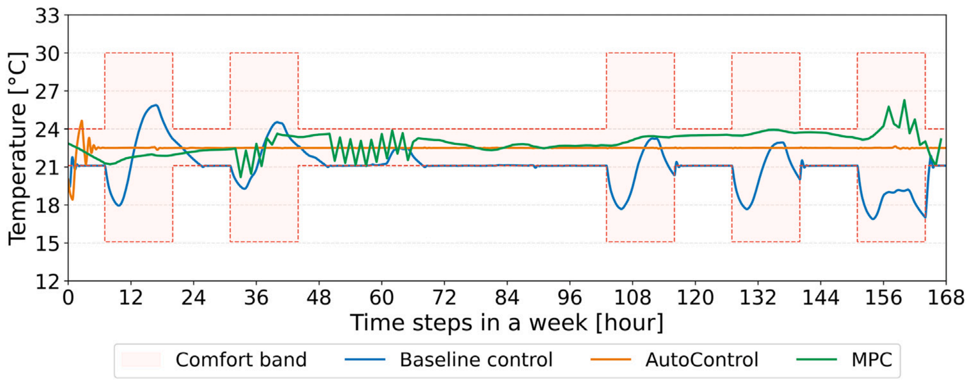


Fig. 12. The comparison of indoor temperature controlled by the baseline controller, MPC and AutoControl during a week for the BESTEST_Hydronic test case.

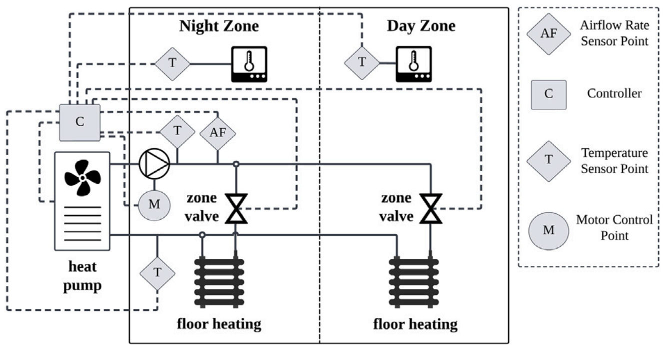


Fig. 13. The overview of the HVAC system for the Two_Zone_Apartment_Hydronic test case.

Table 7
API costs of AutoControl for the first and second LLM-based agent.

	Case 1	Case 2	Case 3
Costs of Agent1 ($)	0.014	0.010	0.029
Costs of Agent2 ($)	0.010	0.009	0.017
Total costs ($)	0.024	0.019	0.046

Table 8
Execution time of AutoControl for each case under single-threaded and multi-threaded configuration.

	Single-thread		Multi-thread	
	PSO optimization	Total time	PSO optimization	Total time
Case 1	32.79 min	32.98 min	4.16 min	4.35 min
Case 2	27.66 min	27.82 min	3.91 min	4.07 min
Case 3	52.88 min	53.23 min	7.40 min	7.75 min

The user only needs to provide basic requirements, such as the temperature set point etc., and the rest of the expertise-embedded process will be completed by the two LLM-based agents. Compared with traditional control design flow, the proposed method liberates the user from the labor-intensive work and reduces technique barriers for implementing controllers in different HVAC systems.

Second, AutoControl has good control performance in all the three test cases with different HVAC systems and control variables, demonstrating superior generalization ability. Additionally, for the complex multiple-zone system, AutoControl demonstrates robust performance by initially identifying the number of zones and subsequently designing a PI controller for each zone. This strategy enables the application of AutoControl to a wide range of HVAC control configurations.

Third, the implementation cost of applying AutoControl to HVAC control problems is low, with an average cost of $0.029 for the entire design process using GPT-4o. This cost-effectiveness characteristic reflects the potential for deploying AutoControl in real-world applications. Fourth, AutoControl is highly modularized, allowing each component within the workflow to be flexibly replaced with alternative algorithms. For instance, the PSO module can be conveniently substituted with rule-based or other heuristic optimization algorithms (e.g., Simulated Annealing, Genetic Algorithm, etc.), leading to high adaptability and flexibility for diverse control tasks.

4.2. Limitations and future work

Despite of the advantages of the AutoControl, there are some limitations in the current work. First, the interpretation of the Brick models may contain semantic errors, which can directly impact on the descriptions of the control and measurement variables. Given that these descriptions are crucial for the control expert agent to accurately comprehend the target HVAC system, where any deficiencies in interpretation quality may lead to poor control performance. Second, the structural information of the HVAC system is hidden within the configurations of the control and measurement variables, making it challenging for the agent to accurately understand, particularly in large and complex systems. The relationships between actuators, such as FCUs and radiators, are not explicitly exposed to the second control expert agent, thereby limiting the ability of AutoControl to effectively manage complex HVAC systems.

While AutoControl has demonstrated superior performance in the multi-zone configuration, as shown in Case 3, its applicability is limited regarding the dynamics of HVAC configurations. Recognition of a

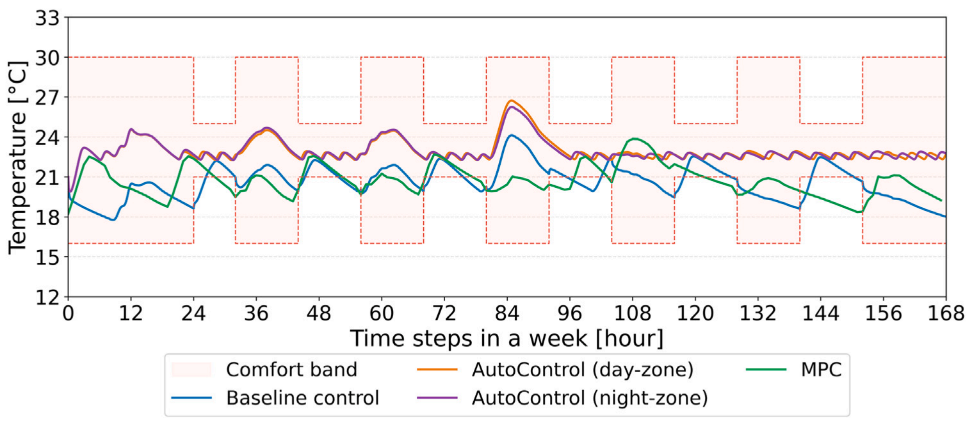


Fig. 14. The comparison of indoor temperature controlled by the baseline controller, MPC and AutoControl during a week for the Two_Zone_Apartment_Hydronic test case.

```python
class Controller:
    def __init__(self, TSet, k_p, k_i):
        self.TSet = TSet
        self.k_p = k_p
        self.k_i = k_i
    def compute_control(self, y, forecasts=None):

        if not hasattr(self, 'integral'):
            self.integral = 0

        # Compute error
        e = self.TSet - y['$measurement_name$']

        # Update integral term
        self.integral += e

        # Compute control value
        value = self.k_p * e + self.k_i * self.integral

        u = {{
            '$control_name1$': value,
            '$control_name2$': $fixed_value$,
        }}

        return u
    def initialize(self):
        u = {{
            '$control_name1$': $fixed_value$,
            '$control_name2$': $fixed_value$,
        }}
        return u
```

Fig. A.1. The example of the code template in the prompt.

multi-zone system is achieved by semantic identification based on variable prefixes. However, the prefixes rules may become quite complex with interconnections between different zones, which requires further investigation and enhancements on the semantic reasoning and control coordination logic. On the other hand, AutoControl faces challenges in changing building geometries (e.g., building retrofit), since the static Brick model is implemented in the current workflow and geometric changes will require updating of the Brick model and re-design of the controller.

In addition, the current implementation of AutoControl assumes a complete and consistent Brick Schema pattern and does not explicitly address irregular or incomplete semantic models. This fixed-schema design limits the generalization ability of the parsing process and prevents a systematic analysis of potential misclassifications under schema variations. Extending the Agent to support heterogeneous schema structures and conducting sensitivity/error analysis will be important directions for future work.

Finally, a virtual testbed is essential for AutoControl to fine-tune the parameters of the PI controller, because the fine-tuning process in real-world systems may risk damaging HVAC equipment or causing severe indoor environmental discomfort. However, many new buildings lack access to such testbeds, which introduces practical deployment challenges of AutoControl. Additionally, while the semantic model can solve most problems arising from building heterogeneity, control performance may still suffer from missing HVAC data points. These include critical sensor measurements or operational parameters that are unavailable due to sensor failures, communication errors, or incomplete system documentation. On the other hand, AutoControl concentrates on the design phase of the building controller. There are some issues when integrating with existing Building Automation Systems (BAS), which

```
Measurements vars:
[fcu_reaPHea_y] [W] [min=None, max=None]: Measures the thermal power output of the heating system in the
    fan coil unit.
[fcu_reaPCoo_y] [W] [min=None, max=None]: Measures the electric power consumption of the cooling system in
    the fan coil unit.
[fcu_reaPFan_y] [W] [min=None, max=None]: Measures the electric power consumption of the fan in the fan
    coil unit.
[zon_reaPLig_y] [W] [min=None, max=None]: Measures the electric power consumption of the lighting in the
    zone.
[zon_reaPPlu_y] [W] [min=None, max=None]: Measures the electric power consumption of plug loads in the
    zone.
[fcu_reaFloSup_y] [KiloGM-PER-SEC] [min=None, max=None]: Measures the supply air flow rate in the fan coil
    unit.
[zon_reaCO2RooAir_y] [PPM] [min=None, max=None]: Measures the CO2 concentration in the room air of the
    zone.
[zon_reaTRooAir_y] [K] [min=None, max=None]: Measures the air temperature in the room of the zone.

Control vars:
[con_oveTSetCoo_u] [K] [min=278.15, max=308.15]: Sets the cooling temperature setpoint for the zone air.
[con_oveTSetHea_u] [K] [min=278.15, max=308.15]: Sets the heating temperature setpoint for the zone air.
[fcu_oveTSup_u] [K] [min=285.15, max=313.15]: Sets the supply air temperature setpoint for the fan coil
    unit.
[fcu_oveFan_u] [] [min=0.0, max=1.0]: Commands the fan operation in the fan coil unit.
[con_oveTSetCoo_activate] [] [min=None, max=None]: Activation signal for the cooling temperature setpoint
    control; 1 activates, 0 deactivates (default).
[con_oveTSetHea_activate] [] [min=None, max=None]: Activation signal for the heating temperature setpoint
    control; 1 activates, 0 deactivates (default).
[fcu_oveTSup_activate] [] [min=None, max=None]: Activation signal for the supply air temperature setpoint
    control; 1 activates, 0 deactivates (default).
[fcu_oveFan_activate] [] [min=None, max=None]: Activation signal for the fan command control; 1 activates,
    0 deactivates (default).
```

Fig. A.2. The example of generated variable descriptions from the Semantic LLM-based agent for the BESTEST_Air test case.

```
Measurements vars:
reaQHea_y W [min=None, max=None]: This is the sensed value for heating thermal power, measured in watts.
reaPPum_y W [min=None, max=None]: This is the sensed electric power for a pump, measured in watts.
reaCO2RooAir_y PPM [min=None, max=None]: This is the sensed CO2 concentration in the room air, measured in
    parts per million.
reaTRoo_y K [min=None, max=None]: This is the sensed temperature in the room air, measured in Kelvin.

Control vars:
oveTSetSup_u K [min=293.15, max=353.15]: This is the setpoint for hot water supply temperature, measured
    in Kelvin, and typically activated by oveTSetSup_activate.
oveTSetSup_activate [min=0, max=1]: This is the activation signal for the hot water supply temperature
    setpoint control variable oveTSetSup_u (1 activates, 0 deactivates).
oveTSetCoo_u K [min=278.15, max=308.15]: This is the setpoint for the zone air cooling temperature,
    measured in Kelvin, and typically activated by oveTSetCoo_activate.
oveTSetCoo_activate [min=0, max=1]: This is the activation signal for the zone air cooling temperature
    setpoint control variable oveTSetCoo_u (1 activates, 0 deactivates).
oveTSetHea_u K [min=278.15, max=308.15]: This is the setpoint for the zone air heating temperature,
    measured in Kelvin, and typically activated by oveTSetHea_activate.
oveTSetHea_activate [min=0, max=1]: This is the activation signal for the zone air heating temperature
    setpoint control variable oveTSetHea_u (1 activates, 0 deactivates).
ovePum_u [min=0, max=1]: This is the on/off command for a pump (1 turns on, 0 turns off), and typically
    activated by ovePum_activate.
ovePum_activate [min=0, max=1]: This is the activation signal for the on/off command control variable
    ovePum_u (1 activates, 0 deactivates).
```

Fig. A.3. The example of generated variable descriptions from the Semantic LLM-based agent for the BESTEST_Hydronic test case.

```
Measurements vars:
dayZon_reaPowQint_y W [min=None, max=None]: This is the sensor measurement for the heat power in the
    day zone.
nigZon_reaPowQint_y W [min=None, max=None]: This is the sensor measurement for the heat power in the
    night zone.
dayZon_reaPowFlooHea_y W [min=None, max=None]: This is the sensor measurement for the floor heating
    power in the day zone.
nigZon_reaPowFlooHea_y W [min=None, max=None]: This is the sensor measurement for the floor heating
    power in the night zone.
dayZon_reaPLig_y W [min=None, max=None]: This is the sensor measurement for the lighting electric
    power in the day zone.
dayZon_reaPPlu_y W [min=None, max=None]: This is the sensor measurement for the electric power in the
    day zone.
hydronicSystem_reaPPum_y W [min=None, max=None]: This is the sensor measurement for the pump electric
    power in the hydronic system.
hydronicSystem_reaPeleHeaPum_y W [min=None, max=None]: This is the sensor measurement for the
    electric heating power of the pump in the hydronic system.
nigZon_reaPLig_y W [min=None, max=None]: This is the sensor measurement for the lighting electric
    power in the night zone.
nigZon_reaPPlu_y W [min=None, max=None]: This is the sensor measurement for the electric power in the
    night zone.
dayZon_reaMFloHea_y KiloGM-PER-SEC [min=None, max=None]: This is the sensor measurement for the hot
    water flow rate in the day zone.
nigZon_reaMFloHea_y KiloGM-PER-SEC [min=None, max=None]: This is the sensor measurement for the hot
    water flow rate in the night zone.
dayZon_reaCO2RooAir_y PPM [min=None, max=None]: This is the sensor measurement for CO2 levels in the
    room air in the day zone.
nigZon_reaCO2RooAir_y PPM [min=None, max=None]: This is the sensor measurement for CO2 levels in the
    room air in the night zone.
dayZon_reaTavgFloHea_y K [min=None, max=None]: This is the sensor measurement for the average floor
    heating temperature in the day zone.
nigZon_reaTavgFloHea_y K [min=None, max=None]: This is the sensor measurement for the average floor
    heating temperature in the night zone.
dayZon_reaTRooAir_y K [min=None, max=None]: This is the sensor measurement for the room air
    temperature in the day zone.
nigZon_reaTRooAir_y K [min=None, max=None]: This is the sensor measurement for the room air
    temperature in the night zone.
hydronicSystem_reaTretFloHea_y K [min=None, max=None]: This is the sensor measurement for the
    returning floor heating water temperature in the hydronic system.
dayZon_reaTsupFloHea_y K [min=None, max=None]: This is the sensor measurement for the supply floor
    heating water temperature in the day zone.
nigZon_reaTsupFloHea_y K [min=None, max=None]: This is the sensor measurement for the supply floor
    heating water temperature in the night zone.
dayZon_reaTretFloHea_y K [min=None, max=None]: This is the sensor measurement for the returning floor
    heating water temperature in the day zone.
nigZon_reaTretFloHea_y K [min=None, max=None]: This is the sensor measurement for the returning floor
    heating water temperature in the night zone.

Control vars:
hydronicSystem_oveMpumCon_u KiloGM-PER-SEC [min=0.0, max=5.0]: This is the override control setpoint
    for the hot water flow rate in the pump of the hydronic system, typically activated by
    hydronicSystem_oveMpumCon_activate.
hydronicSystem_oveMpumCon_activate [min=0, max=1]: This is the activation signal for the
    hydronicSystem_oveMpumCon_u control variable (1 activates, 0 deactivates).
hydronicSystem_oveTHea_u K [min=273.15, max=318.15]: This is the override control setpoint for the
    hot water temperature in the hydronic system, typically activated by
    hydronicSystem_oveTHea_activate.
hydronicSystem_oveTHea_activate [min=0, max=1]: This is the activation signal for the
    hydronicSystem_oveTHea_u control variable (1 activates, 0 deactivates).
```

Fig. A.4. The first part of the example of generated variable descriptions from the Semantic LLM-based agent for the Two_Zone_Apartment_Hydronic test case.

```
thermostatDayZon_oveTsetZon_u K [min=273.15, max=318.15]: This is the override control setpoint for
    the air temperature in the day zone, typically activated by thermostatDayZon_oveTsetZon_activate.
thermostatDayZon_oveTsetZon_activate [min=0, max=1]: This is the activation signal for the
    thermostatDayZon_oveTsetZon_u control variable (1 activates, 0 deactivates).
thermostatNigZon_oveTsetZon_u K [min=273.15, max=318.15]: This is the override control setpoint for
    the air temperature in the night zone, typically activated by
    thermostatNigZon_oveTsetZon_activate.
thermostatNigZon_oveTsetZon_activate [min=0, max=1]: This is the activation signal for the
    thermostatNigZon_oveTsetZon_u control variable (1 activates, 0 deactivates).
hydronicSystem_oveMDayZ_u [min=0.0, max=1.0]: This is the override command for the valve position in
    the hydronic system for the day zone, typically activated by hydronicSystem_oveMDayZ_activate.
hydronicSystem_oveMDayZ_activate [min=0, max=1]: This is the activation signal for the
    hydronicSystem_oveMDayZ_u control variable (1 activates, 0 deactivates).
hydronicSystem_oveMNigZ_u [min=0.0, max=1.0]: This is the override command for the valve position in
    the hydronic system for the night zone, typically activated by hydronicSystem_oveMNigZ_activate.
hydronicSystem_oveMNigZ_activate [min=0, max=1]: This is the activation signal for the
    hydronicSystem_oveMNigZ_u control variable (1 activates, 0 deactivates).
```

Fig. A.5. The second part of the example of generated variable descriptions from the Semantic LLM-based agent for the Two_Zone_Apartment_Hydronic test case.

requires additional adaptation to bridge the generated controller with real-world systems.

In future work, the performance of AutoControl will be enhanced through the following key improvements. First, a more robust and explicit methodology will be developed to interpret critical information from the Brick model for the second control expert agent to deal with the large and complex HVAC systems. Second, advanced control strategies, such as Model Predictive Control (MPC) and Reinforcement Learning (RL), will be integrated into AutoControl to enhance its control performance and effectively address the challenges of dealing with the uncertainty and dynamics of HVAC systems. Third, energy efficiency will be considered in the controller design to help AutoCotrol better align with the widely recognized standard (i.e., ASHRAE Standard 90.1). Fourth, a fault-tolerant and robust control strategy will be included in the controller design to enhance the robustness of the HVAC system regarding long-term working scenarios and high-dynamic occupancy conditions. Finally, the development of a BAS middleware will be considered to integrate AutoControl with standard building automation protocols. Such integrations will facilitate the deployment of auto-generated controllers onto existing BAS framework.

5. Conclusions

Efficient control of building energy systems is crucial for maintaining comfortable environments and reducing carbon emissions. However, reusing efficient controllers across different buildings is challenging as no two buildings are similar. As a result, significant customization efforts are needed to design an efficient controller for a target building, which is a process that requires substantial labor and expertise. In this study, we proposed AutoControl, an end-to-end fully automated workflow based on Large Language Models (LLMs). AutoControl generates Proportional-Integral (PI) controllers for building HVAC systems using Brick models as inputs, eliminating the need for human intervention. The method has two LLM-based agents: the first agent interprets HVAC system configuration information from the Brick model, while the second agent generates the controller codes based on the interpreted configurations and user specified requirements. In the final stage, Particle Swarm Optimization (PSO) is employed to fine-tune the parameters of the PI controller to achieve satisfactory control performance.

Experiments have been conducted on three distinct test cases using the BOPTest virtual testbed. The selected HVAC systems includes the Fan Coil Unit (FCU) system, the Hydronic radiator system, and the Hydronic heat pump system, encompassing both single-zone and multi-zone configurations. AutoControl has demonstrated robust temperature control performance by maintaining zone temperatures close to the set points, achieving an average Mean Absolute Error (MAE) of 0.323 °C and Root Mean Square Error (RMSE) of 0.766 °C over a one-week testing period. Additionally, the total costs of this process is $0.029 based on the current GPT-4o pricing, highlighting its potential for commercial deployment in practical applications. In general, AutoControl has four key advantages: a high level of automation, low cost, robust control performance, and a well-modularized design. However, it remains challenges in dealing with large and complex HVAC systems due to limitations in semantic interpretation from Brick models and the limited ability of PI controllers.

CRediT authorship contribution statement

Ziqi Hu: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Conceptualization. **Mingchen Li:** Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Conceptualization. **Hao Tang:** Writing – review & editing, Investigation. **Zhe Wang:** Writing – review & editing, Supervision, Project administration, Investigation, Funding acquisition, Conceptualization.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used DeepSeek Chat in order to check grammar and language. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Declaration of competing interest

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

Acknowledgments

This work is supported by the National Science Fund for Excellent Young Scholars (Grant number 52322813), and by 2023 HKUST Shenzhen-Hong Kong Collaborative Innovation Institute Green Sustainability Special Fund, from Shui On Xintiandi and the InnoSpace GBA.

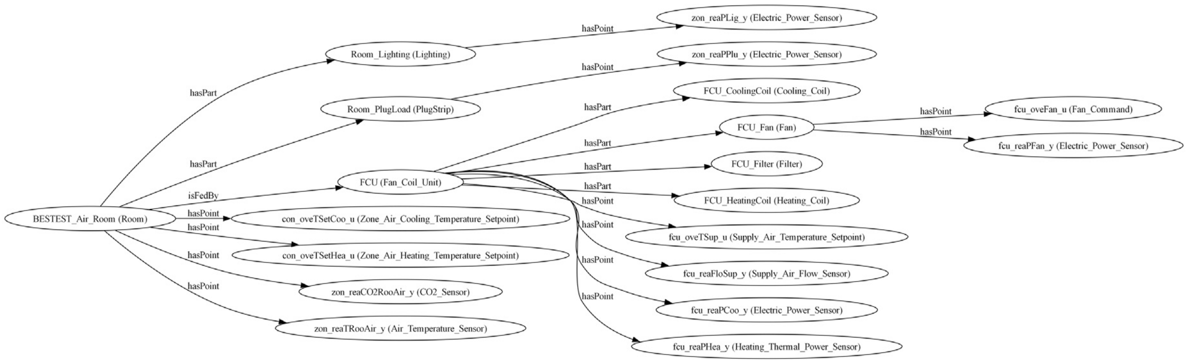

BESTEST_Air_Room (Room)
hasPart
Room_Lighting (Lighting)
hasPoint
zon_reaPLig_y (Electric_Power_Sensor)
hasPart
Room_PlugLoad (PlugStrip)
hasPoint
zon_reaPPlu_y (Electric_Power_Sensor)
isFedBy
FCU (Fan_Coil_Unit)
hasPart
FCU_CoolingCoil (Cooling_Coil)
hasPart
FCU_Fan (Fan)
hasPoint
fcu_oveFan_u (Fan_Command)
hasPoint
fcu_reaPFan_y (Electric_Power_Sensor)
hasPart
FCU_Filter (Filter)
hasPart
FCU_HeatingCoil (Heating_Coil)
hasPoint
fcu_oveTSup_u (Supply_Air_Temperature_Setpoint)
hasPoint
fcu_reaFloSup_y (Supply_Air_Flow_Sensor)
hasPoint
fcu_reaPCoo_y (Electric_Power_Sensor)
hasPoint
fcu_reaPHea_y (Heating_Thermal_Power_Sensor)
hasPoint
con_oveTSetCoo_u (Zone_Air_Cooling_Temperature_Setpoint)
hasPoint
con_oveTSetHea_u (Zone_Air_Heating_Temperature_Setpoint)
hasPoint
zon_reaCO2RooAir_y (CO2_Sensor)
hasPoint
zon_reaTRooAir_y (Air_Temperature_Sensor)

Fig. A.6. Brick semantic model for the BESTEST_Air test case.

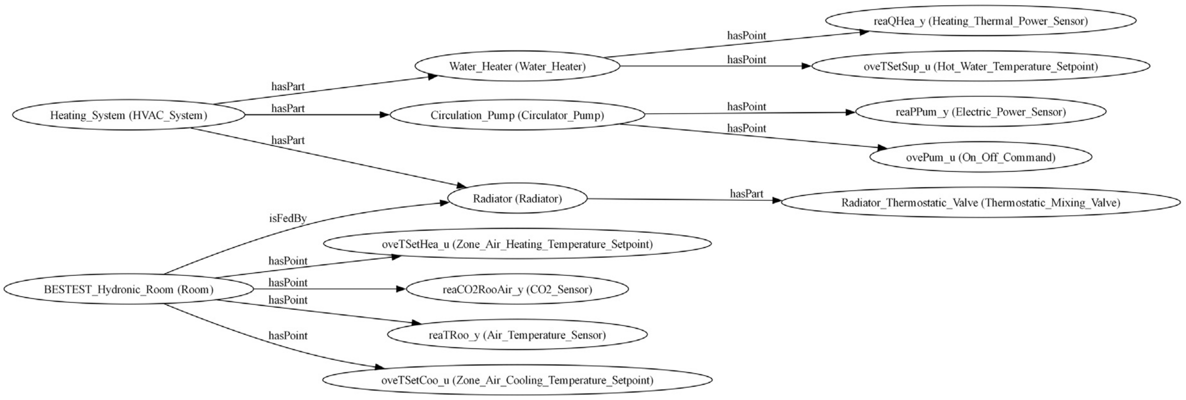

Heating_System (HVAC_System)
hasPart
Water_Heater (Water_Heater)
hasPoint
reaQHea_y (Heating_Thermal_Power_Sensor)
hasPoint
oveTSetSup_u (Hot_Water_Temperature_Setpoint)
hasPart
Circulation_Pump (Circulator_Pump)
hasPoint
reaPPum_y (Electric_Power_Sensor)
hasPoint
ovePum_u (On_Off_Command)
hasPart
Radiator (Radiator)
hasPart
Radiator_Thermostatic_Valve (Thermostatic_Mixing_Valve)
BESTEST_Hydronic_Room (Room)
isFedBy
hasPoint
oveTSetHea_u (Zone_Air_Heating_Temperature_Setpoint)
hasPoint
reaCO2RooAir_y (CO2_Sensor)
hasPoint
reaTRoo_y (Air_Temperature_Sensor)
hasPoint
oveTSetCoo_u (Zone_Air_Cooling_Temperature_Setpoint)

Fig. A.7. Brick semantic model for the BESTEST_Hydronic test case.

Appendix

A.1. The example of the code template

The example code template is illustrated in Fig. A.1.

A.2. The definitions of the errors in objective function of PSO

The definitions of $e_{\text{avg\_error}}$ and $e_{\text{over\_shoot}}$ in the objective function of PSO are given by:

$$e_{\text{avg}} = \frac{1}{N}\sum_{t=1}^{N} |T_t - T_{\text{setpoint}}|, \tag{A.1}$$

$$e_{\text{over\_shoot}} = \max\left(0, \frac{T_{\max} - T_{\text{setpoint}}}{T_{\text{setpoint}} - T_0}\right), \tag{A.2}$$

where, N is the number of control steps, T_t is the zone temperature at time step t, T_{setpoint} is the set point of the zone temperature, $T_{\max}$ is the maximum temperature during the control horizon, T_0 is the initial zone temperature, $t_{\max}$ is a hyper-parameter which is set to 500 in this study. $e_{\text{settling\_time}}$ is the time required for the zone temperature to enter and remain within the comfort band.

A.3. The definitions of evaluation metrics

The definitions of MAE, RMSE and thermal discomfort ΔT_d are given by:

$$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n} |\, T_{\text{real}} - T_{\text{set}} \,|, \tag{A.3}$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n} (T_{\text{real}} - T_{\text{set}})^2}, \tag{A.4}$$

$$\Delta T_d = \frac{1}{N}\sum_{z=1}^{N}\int_{t_s}^{t_e}\Big(\max\left(T_{\text{real}}(t) - T_l(t), 0\right) + \max\left(T_u(t) - T_{\text{real}}(t), 0\right)\Big)\, dt, \tag{A.5}$$

where, n is the number of control steps; T_{real} is the real zone temperature provided by BOPTest; T_{set} is the set point of the zone temperature; T_l and T_u are the lower and upper comfort bounds of the zone, respectively; N is the number of thermal zones; and t_s and t_e are the start step and end of control horizon, respectively. Thermal discomfort [Kh/zone] represents the time-integrated temperature deviation beyond predefined comfort boundaries. ΔT_d provides a comprehensive evaluation of thermal comfort by simultaneously accounting for both the magnitude and duration of thermal discomfort, enabling comprehensive evaluation of building thermal performance.

A.4. The formulation of model predictive control

The Model Predictive Control (MPC) method implemented in this paper is RC-model based MPC, which utilizes a 3R2C model to simulate the thermal dynamics of the operation zone. There are two models that need to be fitted before running the optimization process, which are the thermal model and the actuator model, respectively. The thermal model is formulated based on 3R2C model, and the ordinary differential equations are defined as

$$C_e\frac{dT_e}{dt} = \frac{T_a - T_e}{R_{ea}} + \frac{T_o - T_{oe}}{R_{ie}} + A_e I_{sol} \tag{A.6}$$

$$C_i\frac{dT_i}{dt} = \frac{T_a - T_i}{R_{win}} + \frac{T_e - T_i}{R_{ie}} + \dot{Q}_h + K\dot{Q}_{int} + A_i I_{sol} \tag{A.7}$$

where, C_e is heat capacity of the external wall, C_i is the heat capacity of indoor environment, T_a, T_e and T_i are the temperature of the outdoor, external wall and indoor, respectively, R_{win} is the thermal resistance

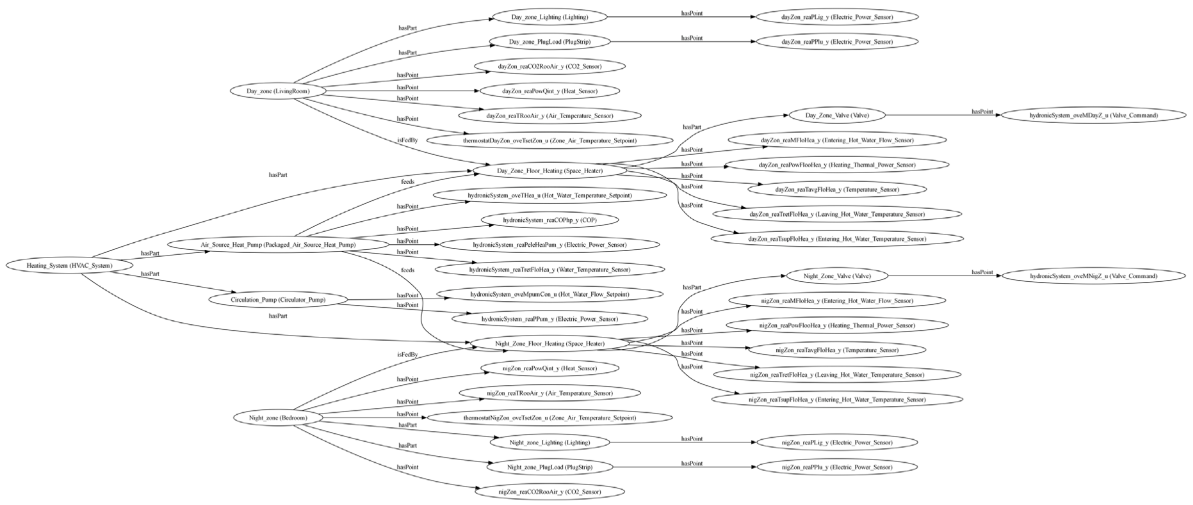

Heating_System (HVAC_System)
Air_Source_Heat_Pump (Packaged_Air_Source_Heat_Pump)
Circulation_Pump (Circulator_Pump)
Day_zone (LivingRoom)
Night_zone (Bedroom)
Day_zone_Lighting (Lighting)
Day_zone_PlugLoad (PlugStrip)
dayZon_reaCO2RooAir_y (CO2_Sensor)
dayZon_reaPowQint_y (Heat_Sensor)
dayZon_reaTRooAir_y (Air_Temperature_Sensor)
thermostatDayZon_oveTsetZon_u (Zone_Air_Temperature_Setpoint)
Day_Zone_Floor_Heating (Space_Heater)
hydronicSystem_oveTHea_u (Hot_Water_Temperature_Setpoint)
hydronicSystem_reaCOPhp_y (COP)
hydronicSystem_reaPeleHeaPum_y (Electric_Power_Sensor)
hydronicSystem_reaTretFloHea_y (Water_Temperature_Sensor)
hydronicSystem_oveMpumCon_u (Hot_Water_Flow_Setpoint)
hydronicSystem_reaPPum_y (Electric_Power_Sensor)
Night_Zone_Floor_Heating (Space_Heater)
nigZon_reaPowQint_y (Heat_Sensor)
nigZon_reaTRooAir_y (Air_Temperature_Sensor)
thermostatNigZon_oveTsetZon_u (Zone_Air_Temperature_Setpoint)
Night_zone_Lighting (Lighting)
Night_zone_PlugLoad (PlugStrip)
nigZon_reaCO2RooAir_y (CO2_Sensor)
dayZon_reaPLig_y (Electric_Power_Sensor)
dayZon_reaPPlu_y (Electric_Power_Sensor)
Day_Zone_Valve (Valve)
hydronicSystem_oveMDayZ_u (Valve_Command)
dayZon_reaMFloHea_y (Entering_Hot_Water_Flow_Sensor)
dayZon_reaPowFlooHea_y (Heating_Thermal_Power_Sensor)
dayZon_reaTavgFloHea_y (Temperature_Sensor)
dayZon_reaTretFloHea_y (Leaving_Hot_Water_Temperature_Sensor)
dayZon_reaTsupFloHea_y (Entering_Hot_Water_Temperature_Sensor)
Night_Zone_Valve (Valve)
hydronicSystem_oveMNigZ_u (Valve_Command)
nigZon_reaMFloHea_y (Entering_Hot_Water_Flow_Sensor)
nigZon_reaPowFlooHea_y (Heating_Thermal_Power_Sensor)
nigZon_reaTavgFloHea_y (Temperature_Sensor)
nigZon_reaTretFloHea_y (Leaving_Hot_Water_Temperature_Sensor)
nigZon_reaTsupFloHea_y (Entering_Hot_Water_Temperature_Sensor)
nigZon_reaPLig_y (Electric_Power_Sensor)
nigZon_reaPPlu_y (Electric_Power_Sensor)
hasPart
hasPoint
isFedBy
feeds

Fig. A.8. Brick semantic model for the Two_Zone_Apartment_Hydronic test case.

between the window and the interior node, R_{ea} is the thermal resistance between the outdoor and the external wall, R_{ie} is the thermal resistance between the external wall and the interior node, $\dot{Q}_{int}$ is the radiative internal gains of the zone, $\dot{Q}_h$ is the thermal power released to the zone from the actuator, I_{sol} is the global solar irradiation, A_i and A_e are the effective window area corresponding to node T_i and T_e, respectively, and K is the scaling factor to estimate total internal heat gains from $\dot{Q}_{int}$. The relationship between $\dot{Q}_h$ and the input signal is characterized by the actuator model, which is typically represented by a linear equation, which is defined as $\dot{Q}_h = a \cdot u + b$, where a and b are the coefficients, u is the input signal provided by the actuator.

MPC is an optimization-based approach that utilizes receding horizon strategy to predict system states and generate optimal action sequences that minimize a specific objective function. Meanwhile, the optimization also needs to be conducted under certain constraints. The objective function and the constraints implemented in this paper are defined as

$$\min_{u \in \mathbb{R}^m} \sum_{k=n}^{n+m} \left(\tilde{T}_i^k - T_{ref}^k \right)^2 \tag{A.8}$$

s.t.

$$\widetilde{T}_i^{n-1} = T_i^{n-1} \tag{A.9}$$

$$\widetilde{T}_i^{k+1} = f\left(\widetilde{T}_i^k, T_a^k, I_{\mathrm{sol}}^k, \dot{Q}_h^k, \dot{Q}_{\mathrm{int}}^k \right), \quad k = n-1, \ldots, n+m-1 \tag{A.10}$$

$$\dot{Q}_h^k = a \cdot u^k + b, \quad k = n, \ldots, n+m \tag{A.11}$$

$$T_{ref}^k = 0.5 \cdot (T_l^k + T_u^k), \quad k = n, \ldots, n+m \tag{A.12}$$

where, $\tilde{T}_i^k$ and T_{ref}^k represent the estimate indoor temperature and reference temperature at time step k, respectively, T_l^k and T_u^k is the upper and lower bound temperature at time step k.

A.5. The properties of the three test cases

The properties of the three test cases, simulation setup and the control variable of each test case that been utilized by AutoControl are illustrated in Table A.1. The property information includes architectural parameters (i.e., location, floor area, height and window area), occupancy schedules, and HVAC setup. The simulation setup includes start day of each test case, simulation step, simulation length and whether to use warm-up. The control variable is automatically selected by expert LLM agent based on semantic information.

A.6. Sample outputs of the semantic agent for the three test cases

Figs. A.2–A.5 demonstrate sample outputs of the first semantic agent for the three BOPTEST test cases: *BESTEST_Air*, *BESTEST_Hydronic*, and *Two_Zone_Apartment_Hydronic*. The outputs integrate essential metadata related to the measurement and control variables of the system, including:

- Variable name,
- Unit of measurement,
- Operational boundaries (min/max limits),
- Descriptive definition.

The structured representation enables unambiguous identification of measurable states and controllable actuators, which provide critical prior knowledge for the second expert agent to generate PI controller codes.

A.7. Semantic models of the three test cases

Figs. A.6–A.8 present the Brick-based semantic models for the three BOPTEST test cases: *BESTEST_Air*, *BESTEST_Hydronic*, and *Two_Zone_Apartment_Hydronic*. These models formalize the relationships between physical spaces (e.g., rooms), HVAC components (e.g., fan coil units, heating/cooling coils), and their associated control and sensing points. The models capture not only the equipment composition (e.g., parts of an FCU), but also functional connections such as which system feeds which space, and which points are used to monitor or control them.

These semantic models form the basis for downstream reasoning in AutoControl. Specifically, Agent 1 uses these graphs to (1) identify relevant variables — such as setpoints and commands — by executing pre-defined SPARQL queries (see Fig. 3); and (2) generate human-readable descriptions for each variable, including its function, type, and unit. The resulting structured and annotated variable set is passed to Agent 2, which uses it to generate executable control code tailored to each test case.

Data availability

No data was used for the research described in the article.

Table A.1
Properties of the three test cases, simulation setup and the control variable of each test case.

Category	Specification	Value		
		BESTEST_Air	BESTEST_Hydronic	Two_Zone_Apartment_Hydronic
Building	Location	Denver	Brussels	Milan
	Floor area	6 × 8 m^2	6 × 8 m^2	Night zone: 17.97 m^2, Day zone: 22.02 m^2
	Height	2.7 m	2.7 m	2.7 m
	Window area	12 m^2	12 m^2	Night zone: 2.35 × 1.6 m^2, Day zone: 2.35 × 2.5 m^2
Occupancy	Occupants	2	1	2
	Weekdays	08:00–18:00	20:00–07:00	20:00–08:00
	Weekends	08:00–18:00	00:00–24:00	00:00–00:00
HVAC	Equipment	Fan coil unit	Water heat pump	Air source heat pump
	Comfort range	Occupied: 21–24 °C, Unoccupied: 15–30 °C	Occupied: 21–24 °C, Unoccupied: 15–30 °C	Occupied: 21–24 °C, Unoccupied: 16–30 °C
Simulation	Start day	282	340	315
	Step	300 s	300 s	300 s
	Length	7 days	7 days	7 days
	If warm-up	No	No	No
Control variable	Name	fcu_oveTSup_u	oveTSetSup_u	hydronicSystem_oveTHea_u
	Description	Air supply temperature	Water supply temperature	Heat system supply temperature

References

[1] USEnergy Information Administration. Annual energy outlook 2021. 2021, URL https://www.eia.gov/outlooks/aeo/. [Accessed 8 November 2021].
[2] Friedman H. Wiring the smart grid for energy savings: Integrating buildings to maximize investment. Portland Energy Conservation Inc. (PECI); 2009.
[3] DoE US. Buildings energy data book. Energy Effic Renew Energy Dep 2011;286:1–3.
[4] Katipamula S, Brambley MR. Methods for fault detection, diagnostics, and prognostics for building systems—A review, Part I. HVAC R Res 2005;11(1):3–25.
[5] Shaikh PH, Nor NB, Nallagownden P, Elamvazuthi I, Ibrahim T. Robust stochastic control model for energy and comfort management of buildings. Aust J Basic Appl Sci 2013;7:137–44.
[6] Shaikh PH, Nor NB, Nallagownden P, Elamvazuthi I. Building energy management through a distributed fuzzy inference system. Int J Eng Technol 2013;5(4):3236–42.
[7] de Dear RJ, Brager GS. Thermal comfort in naturally ventilated buildings: revisions to ASHRAE Standard 55. Energy Build 2002;34(6):549–61. http://dx.doi.org/10.1016/S0378-7788(02)00005-1.
[8] Silva AS, Ghisi E, Lamberts R. Performance evaluation of long-term thermal comfort indices in building simulation according to ASHRAE Standard 55. Build Environ 2016;102:95–115. http://dx.doi.org/10.1016/j.buildenv.2016.03.004.
[9] Gwerder M, Boetschi S, Gyalistras D, Sagerschnig C, Sturzenegger D, Smith R, et al. Integrated predictive Rule-Based control of a Swiss Office Building. In: CLIMA 2013: 11th REHVA world congress & 8th int. conf. IAQVEC. Society of Environmental Engineering; 2013, p. 245.
[10] ASHRAE. ASHRAE guideline 36, High-Performance sequences of operation for HVAC systems. 2021, Available: https://www.ashrae.org/news/ashraejournal/guideline-36-2021-what-snew-and-why-it-s-important, Online.
[11] Faulkner CA, Lutes R, Huang S, Zuo W, Vrabie D. Simulation-Based assessment of ASHRAE guideline 36, considering energy performance, indoor air quality, and control stability. Build Env 2023;240:110371.
[12] Inal OB, Charpentier J-F, Deniz C. Hybrid power and propulsion systems for ships: Current status and future challenges. Renew Sustain Energy Rev 2022;156:111965.
[13] Hu J, Shan Y, Guerrero JM, Ioinovici A, Chan KW, Rodriguez J. Model predictive control of microgrids—An overview. Renew Sustain Energy Rev 2021;136:110422.
[14] Yang S, Wan MP, Chen W, Ng BF, Dubey S. Model predictive control with adaptive Machine-Learning-Based model for building energy efficiency and comfort optimization. Appl Energy 2020;271:115147.
[15] Taheri S, Hosseini P, Razban A. Model predictive control of heating, ventilation, and air conditioning (HVAC) systems: A state-of-the-Art review. J Build Eng 2022;60:105067.
[16] Serale G, Fiorentini M, Capozzoli A, Bernardini D, Bemporad A. Model predictive control (MPC) for enhancing building and HVAC system energy efficiency: Problem formulation, applications and opportunities. Energies 2018;11(3):631.
[17] Joe J, Dong J, Munk J, Kuruganti T, Cui B. Virtual storage capability of residential buildings for sustainable smart city via Model-Based predictive control. Sustain Cities Soc 2021;64:102491.
[18] Killian M, Kozek M. Ten questions concerning model predictive control for energy efficient buildings. Build Env 2016;105:403–12.
[19] Weißmann A, Görges D, Lin X. Energy-Optimal adaptive cruise control combining model predictive control and dynamic programming. Control Eng Pract 2018;72:125–37.
[20] Wei T, Wang Y, Zhu Q. Deep reinforcement learning for building HVAC control. In: Proc. 54th annu. des. autom. conf.. 2017, p. 1–6.
[21] Fang X, Gong G, Li G, Chun L, Peng P, Li W, et al. Deep reinforcement learning optimal control strategy for temperature setpoint Real-Time reset in Multi-Zone building HVAC system. Appl Therm Eng 2022;212:118552.
[22] Nguyen AT, Pham DH, Oo BL, Santamouris M, Y. A, Lim BTH. Modelling building HVAC control strategies using a deep reinforcement learning approach. Energy Build 2024;310:114065.
[23] Zhou X, Du H, Sun Y, Ren H, Cui P, Ma Z. A new framework integrating reinforcement learning, a rule-based expert system, and decision tree analysis to improve building energy flexibility. J Build Eng 2023;71:106536. http://dx.doi.org/10.1016/j.jobe.2023.106536.
[24] Wang J, Ke L. LLM-Seg: Bridging image segmentation and large language model reasoning. In: Proc. IEEE/CVF conf. comput. vis. pattern recognit.. 2024, p. 1765–74.
[25] Fathullah Y, Wu C, Lakomkin E, Jia J, Shangguan Y, Li K, et al. Prompting large language models with speech recognition abilities. In: ICASSP 2024 - 2024 IEEE int. conf. acoust. speech signal process.. 2024, p. 13351–5. http://dx.doi.org/10.1109/ICASSP48485.2024.10447605.
[26] Fakhoury S, Naik A, Sakkas G, Chakraborty S, Lahiri SK. LLM-Based Test-Driven interactive code generation: User study and empirical evaluation. IEEE Trans Softw Eng 2024;50(9):2254–68. http://dx.doi.org/10.1109/TSE.2024.3428972.
[27] Zhang J, Zhang C, Lu J, Zhao Y. Domain-Specific large language models for fault diagnosis of heating, ventilation, and air conditioning systems by Labeled-Data-Supervised Fine-Tuning. Appl Energy 2025;377:124378.
[28] Wan H, Zhang J, Chen Y, Xu W, Feng F. Generative AI application for building industry. 2024, arXiv preprint arXiv:2410.01098.
[29] Islam R, Moushi OM. GPT-4o: The cutting-edge advancement in multimodal LLM. Authorea Prepr 2024. http://dx.doi.org/10.36227/techrxiv.171986596.65533294/v1.
[30] Touvron H, Lavril T, Izacard G, Martinet X, M.-A. L, Lacroix T, et al. LLaMA: Open and efficient foundation language models. 2023, arXiv preprint arXiv:2302.13971 URL https://api.semanticscholar.org/CorpusID:257219404.
[31] Guo D, Yang D, Zhang H, Song J, Zhang R, Xu R, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. 2025, arXiv preprint arXiv:2501.12948.
[32] Liu A, Feng B, Xue B, Wang B, Wu B, Lu C, et al. DeepSeek-V3 technical report. 2024, arXiv preprint arXiv:2412.19437.
[33] Zhou X, Liu R, Tian S, Shen X, Yang X, An J, et al. A review of validation methods for building energy modeling programs. In: Build Simul. 16, (11):Springer; 2023, p. 2027–47.
[34] EnergyPlus. EnergyPlus. 2024, n.d. URL https://energyplus.net/. [Accessed 14 January 2024].
[35] Jiang G, Ma Z, Zhang L, J. C. EPlus-LLM: A large language model-based computing platform for automated building energy modeling. Appl Energy 2024;367:123431. http://dx.doi.org/10.1016/j.apenergy.2024.123431.
[36] Chung HW, Hou L, Longpre S, Zoph B, Tay Y, Fedus W, et al. Scaling instruction-finetuned language models. J Mach Learn Res 2024;25(70):1–53.

[37] Elnour M, Himeur Y, Fadli F, Mohammedsherif H, Meskin N, Ahmad AM, et al. Neural network-based model predictive control system for optimizing building automation and management systems of sports facilities. Appl Energy 2022;318:119153. http://dx.doi.org/10.1016/j.apenergy.2022.119153.

[38] Afzal S, Ziapour BM, Shokri A, Shakibi H, Sobhani B. Building energy consumption prediction using multilayer perceptron neural network-assisted models; comparison of different optimization algorithms. Energy 2023;282:128446. http://dx.doi.org/10.1016/j.energy.2023.128446.

[39] Fan C, Lei Y, Sun Y, Piscitelli MS, Chiosa R, Capozzoli A. Data-centric or algorithm-centric: Exploiting the performance of transfer learning for improving building energy predictions in data-scarce context. Energy 2021;240:122775. http://dx.doi.org/10.1016/j.energy.2021.122775.

[40] Zhang C, Zhang J, Zhao Y, Lu J. Automated data-driven building energy load prediction method based on generative pre-trained transformers (GPT). Energy 2025;318:134824. http://dx.doi.org/10.1016/j.energy.2025.134824.

[41] Merabet GH, Essaaidi M, Haddou MB, Qolomany B, Qadir J, Anan M, et al. Intelligent building control systems for thermal comfort and energy-efficiency: A systematic review of artificial intelligence-assisted techniques. Renew Sustain Energy Rev 2021;144:110969. http://dx.doi.org/10.1016/j.rser.2021.110969.

[42] Dong B, Lam KP. A real-time model predictive control for building heating and cooling systems based on the occupancy behavior pattern detection and local weather forecasting. Build Simul 2013;7(1):89–106. http://dx.doi.org/10.1007/s12273-013-0142-7.

[43] Yussuf RO, Asfour OS. Applications of artificial intelligence for energy efficiency throughout the building lifecycle: An overview. Energy Build 2024;305:113903. http://dx.doi.org/10.1016/j.enbuild.2024.113903.

[44] Abdelwanis MI, Elmezain MI. A comprehensive review of hybrid AC/DC networks: insights into system planning, energy management, control, and protection. Neural Comput Appl 2024;36(29):17961–77. http://dx.doi.org/10.1007/s00521-024-10264-5.

[45] Zhang L, Chen Z. Opportunities of applying large language models in building energy sector. Renew Sustain Energy Rev 2025;214:115558. http://dx.doi.org/10.1016/j.rser.2025.115558, URL https://www.sciencedirect.com/science/article/pii/S136403212500231X.

[46] Guo X, Keivan D, Syed U, Qin L, Zhang H, Dullerud G, et al. ControlAgent: Automating control system design via novel integration of LLM Agents and domain expertise. 2024, arXiv preprint arXiv:2410.19811.

[47] Project Haystack. Project Haystack documentation. 2025, https://project-haystack.org/doc/docHaystack/Docs. [Accessed 10 August 2025].

[48] Industrial Digital Twin Association (IDTA). Specification of the Asset Administration Shell Part 1: Metamodel (Version 3.0). Technical Report IDTA-01001-3-0, Industrial Digital Twin Association; 2023.

[49] OPC Foundation. OPC 30270: OPC UA for asset administration shell (Release 1.00). 2021, https://reference.opcfoundation.org/I4AAS/v100/docs/.

[50] Balaji B, Bhattacharya A, Fierro G, Gao J, J. G, Hong D, Johansen A, Koh J, Ploennigs J, Agarwal Y, Bergés M, Culler D, Gupta R, Kjærgaard MB, Srivastava M, Whitehouse K. Brick: Metadata schema for portable smart building applications. Appl Energy 2018;226:1273–92. http://dx.doi.org/10.1016/j.apenergy.2018.02.091.

[51] Sjoerd R, Nikoletta N, Mark VDP. Modelling with AAS and RDF in Industry 4.0. Comput Ind 2023;148:103910. http://dx.doi.org/10.1016/j.compind.2023.103910.

[52] ASHRAE. ASHRAE BACnet committee works with other organizations on New Standard (223P). 2018, https://www.ashrae.org/news/esociety/ashrae-bacnet-committee-works-with-other-organizations-on-new-standard. [Accessed 10 August 2025].

[53] Project Haystack. ASHRAE's BACnet Committee, Project Haystack and Brick Schema collaborating to provide unified semantic modeling. 2018, https://marketing.project-haystack.org/project-haystack-media/press-releases/ashraes-bacnet-committee-project-haystack-and-brick-schema-collaborating-to-provide-unified-data-semantic-modeling-solution. [Press Release Accessed 10 August 2025].

[54] Peter R, Moritz L, Michael M, Marcus F, Tanja O, Dirk M. TEASER: an open tool for urban energy modelling of building stocks. J Build Perform Simul 2018;11(1):84–98.

[55] Gerhard G, Lutz P. CityGML–Interoperable semantic 3D city models. ISPRS J Photogramm Remote Sens 2012;71:12–33.

[56] Niall B, Athanassios C, Stylianos K, David S, Vassiliki L, Anastasios K, T G, Guerra CA. A Multi-Level digital twin for optimising demand response at the local level without compromising the Well-being of consumers. In: The 9th international conference on construction engineering and project management. 2022.

[57] Martin S, Roman O, Wilfried E. A framework for hardware-in-the-loop testing of an integrated architecture. In: IFIP international workshop on software technolgies for embedded and ubiquitous systems. Springer; 2007, p. 159–70.

[58] Blum D, Arroyo J, Huang S, Drgoňa J, Jorissen F, Walnum HT, et al. Building optimization testing framework (BOPTEST) for simulation-based benchmarking of control strategies in buildings. J Build Perform Simul 2021;14(5):586–610.

[59] BrickSchema. BrickSchema documentation. 2021, https://docs.brickschema.org. [Accessed 05 March 2025].

[60] Attaran SM, Yusof R, Selamat H. Short review on HVAC components, mathematical model of HVAC system and different PID controllers. Int Rev Autom Control 2014;7(3):263–70.

[61] Fraisse G, Viardot C, Lafabrie O, Achard G. Development of a simplified and accurate building model based on electrical analogy. Energy Build 2022;34(10):1017–31.

[62] Bahrami M, Mansoorizadeh M, Khotanlou H. Few-shot learning with prompting methods. In: 2023 6th international conference on pattern recognition and image analysis. IPRIA, 2023, p. 1–5. http://dx.doi.org/10.1109/IPRIA59240.2023.10147172.

[63] Fritzson P, Engelson V. Modelica—A unified object-oriented language for system modeling and simulation. In: ECOOP'98—object-oriented programming: 12th European conference Brussels, Belgium, July 20–24, 1998 proceedings 12. Springer; 1998, p. 67–90.

[64] National Renewable Energy Laboratory. NREL/buildingmotif. 2024, URL https://github.com/NREL/BuildingMOTIF.