

How Large Language Models can empower the co-simulation of occupant behavior and thermal comfort: Unleashing new frontiers and possibilities[#]

Yu Li¹, Xi Chen^{2*}, Chong Meng³, Chunmei Guo⁴, Ben M. Chen²

1 Hong Kong Centre for Logistics Robotics, Hong Kong Special Administrative Region of China

2 Department of Mechanical and Automation Engineering, The Chinese University of Hong Kong, Shatin, N.T, Hong Kong Special Administrative Region of China

3 China Academy of Building Research

4 Tianjin Chengjian University
(Corresponding Author: xichen002@cuhk.edu.hk)

ABSTRACT

Various methods have been developed for the co-simulation of occupant behavior and thermal comfort. However, these approaches often present a steep learning curve for beginners due to the complex and labor-intensive coding tasks required for real-time data exchange between multiple simulation tools (e.g., EnergyPlus, MATLAB, and CFD). Recent advancements in large language models (LLMs) have demonstrated promising code generation capabilities, yet their potential in domain-specific co-simulation tasks remains underexplored. To address this gap, this study proposes an LLM-based code generation framework to facilitate the coupling of occupant behavior modeling with thermal comfort simulation. Given the collaborative nature of such coding tasks, Generative Pre-trained Transformers (GPT) serves as multiple roles, including that of an "Analyst", "Coder", and "Debugger", each is responsible for specific subtasks. A comprehensive co-simulation experiment was conducted to evaluate LLMs in terms of "learning and understanding", "imitating and coding", and "testing and debugging". A general workflow is also provided to guide prompt engineering in efficiently tackling such coding tasks. Results indicate that LLMs can accomplish most coding tasks after several rounds of prompting iterations. Simple coding tasks are completed with minimal guidance, while for more advanced tasks, collaboration between experts and LLMs is required to save time. This study also discusses the advantages and limitations of applying LLMs to building co-simulation and highlights their potential in transforming conventional workflows.

Keywords: Large Language Models, task division, code generation, GPT, building co-simulation, prompt engineering

NONMENCLATURE

<i>Abbreviations</i>	
LLM	Large Language Model
GPT	Generative Pre-trained Transformers
OB-TC	Occupant Behavior and Thermal Comfort
BCVTB	Building Controls Virtual Test Bed

1. INTRODUCTION

Thermal comfort is of essential importance for occupant health, work productivity, and well-being. Accurately identifying how occupants behave within the buildings is crucial for effective thermal comfort assessment and management, given the dynamic and adaptive interactions between occupants and the indoor built environment. Understanding the interplay between occupant behavior and thermal comfort is key to the design of occupant-centric building control systems (i.e., smart window and HVAC systems), aiming to balance energy efficiency and occupant satisfaction. With the dramatic development in computer vision, deep learning algorithms, and the increasing accessibility of multi-source occupant behavior data, data-driven occupant behavior modeling approaches have gained widespread application in thermal comfort assessment, owing to their high robustness and authenticity. Furthermore, co-simulation opens a new direction for predicting and optimizing occupant thermal comfort by better considering occupant behavior impacts.

[#] This is a paper for the 11th Applied Energy Symposium: Low Carbon Cities & Urban Energy Systems (CUE2025), July 18-22, 2024, Kitakyushu, Japan.

Recent breakthroughs in large language models (LLMs) [1], such as ChatGPT, LLaMa, Claude, and DeepSeek, have unlocked new frontiers and possibilities in building sectors, including load forecasting and energy modelling, fault detection and diagnosis of HVAC systems, and document analysis and knowledge extraction [2]. LLMs can efficiently process and analyse large-scale data to generate actionable insights, support decision-making, and extract key information from textual sources. Furthermore, LLMs can assist in developing and refining software and modelling tools, ensuring regulatory compliance, and generating customized educational materials. Due to their inherent flexibility and adaptability, latest LLMs are particularly well-suited for real-time data analysis and predictive modelling, and automating routine operations in energy management systems.

Currently, most applications of LLMs in the building domain focus on automated building energy modelling [3], data tagging and spatial context analysis, energy management, monitoring, and operation optimization, fault detection and diagnosis, along with data-driven prediction and forecasting [2]. However, limited trials on building co-simulation are found [4]. Implementing LLMs for occupant behavior and thermal comfort co-simulation (OB-TC co-simulation) presents several key challenges, including real-time synchronization between multiple simulation tools, maintaining data consistency across platforms, resolving conflicts in overlapping parameter domains, and complicated integration process [4]. It is quite difficult for beginners to formulate high-quality LLMs prompts for generating satisfactory codes. To bridge these gaps, this study aims to address the following three questions to verify and enhance LLMs performance in automating OB-TC co-simulation leveraging GPT-4o:

- Can LLMs effectively translate external task requirements into a structured OB-TC co-simulation framework?
- How can LLM prompts functions be used to automatically generate code for OB-TC co-simulation?
- How to guide LLMs in identifying and correcting runtime errors during OB-TC co-simulation?

2. METHODOLOGY

This study proposes a methodological framework to facilitate the co-simulation of occupant behavior and thermal comfort leveraging LLMs. As shown in Figure 1,

the framework comprises three key steps. Firstly, an initial prompt is provided by users, which describes the fundamental requirements on occupant behavior and thermal comfort co-simulation, and then requires the GPT to define the structure of the co-simulation framework. Secondly, multi-agent LLM techniques are used to generate and revise codes for occupant behavior and thermal comfort co-simulation, which consists of a Central LLM Agent and several LLM task agents. The Central LLM Agent communicates with the user, plans sub-tasks, assigns sub-tasks to specialized LLM task agents, aggregates the results from LLM task agents, and eventually send back results to users. Thirdly, the code generated by LLM task agent (Coder) are executed on the co-simulation platform (Building Controls Virtual Test Bed, BCVTB) via APIs, the runtime errors are fed back to another LLM task agent (Debugger) for troubleshooting, this step is iterated until the revised code is satisfactory and works correctly.

2.1 Experiment preparation

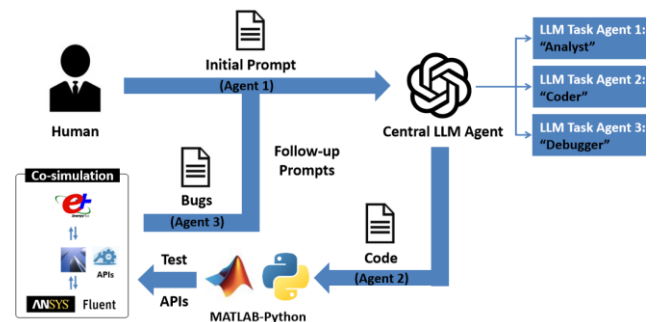


Fig. 1 The methodological framework of LLMs-enabled co-simulation of occupant behavior and thermal comfort.

Due to the labor-intensive process of modeling occupant behavior and predicting thermal comfort, this study starts as a preliminary experiment focused solely on implementing the OB-TC co-simulation workflow, rather than developing complex OB models or TC models from scratch. Therefore, the case models used in this study are selected from previous work [5], which details the development of occupant behavior models and thermal comfort models. This foundational setup allows this work to focus on testing and evaluating LLM-enabled automation in OB-TC co-simulation tasks. The main requirement of this co-simulation experiment is to realize a two-way BES-CFD co-simulation-based window and HVAC control framework for enhancing occupant thermal comfort, as shown in Figure 2.

The occupant behavior and thermal comfort co-simulation tasks are domain-specific and involve multiple

tools such as EnergyPlus, Fluent, and MATLAB/Python via BCVTB (co-simulation platform). To adapt LLMs to generate code for data exchange and interpretation across various simulators, existing EnergyPlus and CFD models in [5] were adopted. Besides, representative examples (rule-based and code-based) for co-simulation in BCVTB were fed to GPT to guide code generation. This process guarantees that domain-relevant syntax, logic structures, and file configurations are followed correctly.

You are a programming expert in co-simulation of occupant behavior and thermal comfort. Please define the structure of the co-simulation framework according to my requirements.

My requirements are described as follows:

The co-simulation of occupant behavior and thermal comfort is domain-specific and involves multiple tools such as EnergyPlus, MATLAB/Python, and Fluent via BCVTB. This study aims to realize a two-way BES-CFD co-simulation-based window and HVAC control framework for enhancing thermal comfort. I have established a EnergyPlus model for simulating the indoor thermal dynamics, a CFD model for simulating indoor airflow/temperature. BCVTB-MATLAB serves as the co-simulation platform to exchange data between EnergyPlus and CFD.

The coupling variables that participated in the exchange between EnergyPlus, Fluent, MATLAB and BCVTB include:

- 1) Interior surface temperatures of walls, roof, floor, windows and skylights, site wind speed and direction, site outdoor air dry bulb temperature, supply air flow rate, supply air temperature, and window-HVAC control signals from EnergyPlus to Fluent, serving as boundary conditions for CFD simulation;
- 2) Zone mean radiant temperature, zone air relative humidity, metabolic rate, and clothing insulation from EnergyPlus to MATLAB, along with air temperature and air velocity at the 48 test points from Fluent to MATLAB for PMV (aPMV) calculation;
- 3) Zone mean air temperature and site outdoor air dry bulb temperature from EnergyPlus to BCVTB, and PMV (aPMV) from MATLAB to BCVTB, serving as decision variables for window-HVAC control; and (4) Window and HVAC control signals at each time step from BCVTB to EnergyPlus.

Fig. 2 Initial Prompting function for defining the structure of the co-simulation framework.

As shown in Figure 3, a multi-agent workflow was adopted where the LLM plays four distinct roles to realize self-collaboration code generation and revision, taking into account the feedback during iterative executions:

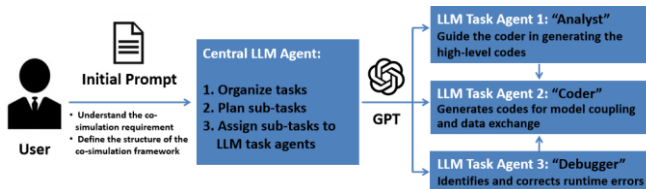


Fig. 3 Architecture of the multi-agent LLMs for code generation.

- **Central LLM Agent:** Responsible for organizing tasks, planning sub-tasks and assigning sub-tasks to LLM task agents (Figure 4).
- **LLM Task Agent 1 (Analyst):** Focus on guiding the Coder in generating the high-level codes, based on the representative examples, i.e., rule-based and code-based examples (Figure 5).

- **LLM Task Agent 2 (Coder):** Generates code files for model coupling and data exchange, based on tailored prompts and example templates.
- **LLM Task Agent 3 (Debugger):** Identifies and corrects runtime errors by analyzing runtime feedback from the simulation environment.

By understanding the structured definition of the co-simulation framework for occupant behavior and thermal comfort provided. Assume that you are the "Central LLM Agent" in a multi-agent LLMs system. The Central LLM Agent is tasked with generating and modifying code for occupant behavior and thermal comfort co-simulation by coordinating three specialized task agents. Please organize tasks, plan sub-tasks, and assign them to three LLM task agents—"Analyst", "Coder", and "Debugger"—responsible for:

1. Analyst: Guiding the coder in generating the high-level codes, based on the representative examples, i.e., rule-based and code-based;
2. Coder: Generates code files for model coupling and data exchange, based on tailored prompts and example templates;
3. Debugger: Identifies and corrects runtime errors by analyzing feedback from the simulation environment.

Fig. 4 Prompts of the Central LLM Agent.

2.2 Analyst: Extracting external knowledge for code generation

In the first stage, after receiving the structure of the co-simulation framework generated by the "Central LLM Agent", the "Analyst" agent then analyses the corresponding sub-tasks, and prepares for the code generation by extracting domain-relevant syntax, logic structures, and data configurations from representative examples provided by user. Two kinds of external knowledge were considered in this case study, namely, rule-based external knowledge and code-based external knowledge, which were used to improve the code correctness of GPT by inserting additional information into a prompt. Rule-based external knowledge can guide GPT to generate codes in an example format for tailored cases. While code-based external knowledge can teach GPT to write correct codes in some cases that GPT is not familiar with. Due to page limit, we only show one sample of each, as displayed in Figure 5.

2.3 Coder: Code generation for OB-TC co-simulation via prompting functions

In the second stage, the LLM plays the role of Coder, executing automated code generation tasks through well-structured prompts and iterative User-GPT interaction. To enhance performance, we employed a multi-sub-agent strategy, where each LLM sub-agent is responsible for specific code components:

BCVTB file generator: Generates BCVTB configuration files, including: co-simulation framework file (.xml), variable configuration file (.cfg), and window-HVAC control logic file (.m).

MATLAB file generator: Handles MATLAB-based components, including:

- main.m: Orchestrates model logic, data exchange and storage.
- runCFDSimulation.m: Calls Fluent simulation.
- calculateThermalComfort.m: Computes thermal comfort indices (e.g., PMV, aPMV).

- writeCFDJournalFile.jou: Prepares journal file for Fluent automation.
- exchangeDoublesWithSocket.m: Exchanges EnergyPlus/MATLAB data with BCVTB.

2.4 Debugger: Self-correction strategy for generated codes

The third stage tasks the LLM with debugging and correction. Once the code is generated and executed, error messages or failed runs are returned as feedback to LLM Task Agent 3 (Debugger). The debugger will translate the error message (e.g., syntax errors, file not found, runtime conflicts, etc), adjust faulty logic or syntax based on domain rules; and finally troubleshoot the code bugs. This step iterates until the simulation runs successfully. This iterative self-correction mimics a user-expert collaboration and evaluates the model's ability to improve code based on simulated runtime responses.

3. RESULTS AND DISCUSSION

3.1 Evaluation of LLMs in case analysis and task divisions

Through the case study, we found that the LLM agent acting as "Analyst" successfully understood our proposed requirements of OB-TC co-simulation, and effectively structured the overall architecture of OB-TC co-simulation. It recognized the appropriate simulation tools and their respective roles in the co-simulation system, divided the workflow into coherent subtasks, and organized data exchange formats and flow in a logical sequence. Notably, the LLM was able to accurately categorize simulation components into distinct modeling environments, such as EnergyPlus, MATLAB, Fluent, and BCVTB. By providing more detailed requirements for OB-TC co-simulation, LLM demonstrated strong capability in interpreting complex co-simulation goals and decomposing them into well-defined components, even in domain-specific building co-simulation applications. After several iterative user-LLMs interactions, Central LLM Agent successfully organized and assigned tasks to three LLM Task Agents, enabling effective code generation and maintenance for OB-TC co-simulation.

3.2 LLMs performance in automated code generation with prompt engineering

In the case of generating the variable configuration file (variables.cfg), as illustrated in the Figure 6, the first-turn code generated based on the rule-based external knowledge largely met the requirements, with only

You are the "Analyst" agent in a multi-agent LLMs system, specializing in learning and extracting the domain-relevant syntax, logic structures, and data configurations from example files. Next is an example xml file (variables.cfg, from BCVTB Manual) that defines the mapping between EnergyPlus and BCVTB variables. Please help summarize its syntax and provide an example template for my case.

Example xml file (refer to <https://simulationresearch.lbl.gov/bcvtb/releases/latest/doc/manual/index.xhtml>):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE BCVTB-variables SYSTEM "variables.dtd">
<BCVTB-variables>
<!-- The next two elements send the set points to E+ -->
<variable source="Ptolemy">
<EnergyPlus schedule="TSetHea"/>
</variable>
<variable source="Ptolemy">
<EnergyPlus schedule="TSetCoo"/>
</variable>
<!-- The next two elements receive the outdoor and
the zone air temperature from E+ -->
<variable source="EnergyPlus">
<EnergyPlus name="ENVIRONMENT" type="OUTDOOR DRY BULB"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="ZSF1" type="ZONE/SYS AIR TEMPERATURE"/>
</variable>
<!-- The next two elements receive the schedule value as an output from E+ -->
<variable source="EnergyPlus">
<EnergyPlus name="TSetHea" type="Schedule Value"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="TSetCoo" type="Schedule Value"/>
</variable>
</BCVTB-variables>
```

(a)

You are the "Analyst" agent in a multi-agent LLMs system, specializing in learning and extracting the domain-relevant syntax, logic structures, and data configurations from example files. Next is an example matlab model that illustrates the implementation of a simulation program written in Matlab that communicates with Ptolemy II through BSD sockets. The simulation program computes the temperature change in two rooms with different capacity. Input to the simulation program are the control signal u_k . Output of the simulation program are the new room temperatures T_{k+1} . The control action is computed in Ptolemy II. Please summarize each part of the example code, including its syntax, external functions and provide an example template for my case.

Example matlab file (refer to <https://github.com/lbl-srg/bcvtb/blob/master/examples/matlab-room/simulateAndExit.m>):

```
% Initialize model variables
delTim = 60; % time step
TIni = 10;
tau = 2*3600;
Q0Hea = 100;
UA = Q0Hea / 20;
TOut = 5;
C = [tau*UA 2*tau*UA];
TRoo = [TIni TIni];
u = [0 0];
% Initialize flags
retVal = 0;
flaWri = 0;
flaRea = 0;
simTimWri = 0;
simTimRea = 0;
% Add path to BCVTB matlab libraries
addpath('strcat(getenv(''BCVTB_HOME''), '/lib/matlab/');
% Establish the socket connection
sockfd = establishClientSocket('socket.cfg');
if sockfd < 0
    fprintf('Error: Failed to obtain socket file descriptor. sockfd=%d\n', ...
        sockfd);
    exit;
end
% Loop for simulation time steps.
simulate=true;
while (simulate)
    % Assign values to be exchanged
    try
        [retVal, flaRea, simTimRea, u] = ...
            exchangeDoublesWithSocket(sockfd, flaWri, length(u), simTimWri, ...
                v);
    catch ME1
        % exchangeDoublesWithSocket had an error. Terminate the connection
        processError(ME1, sockfd, -1);
        simulate = false;
    end
    ... (continued)
```

(b)

Fig. 5 Prompts of the Analyst Agent based on: (a) Rule-based external knowledge; (b) Code-based external knowledge.

minor issues related to name and type. These errors were easily correctable through manual revision. The generated code demonstrated clear logic and was well-structured in accordance with the BCVTB manual.

In the case of generating the MATLAB file (main.m), as illustrated in the Figure 7, LLM can easily summarize

the overall architecture, each section was filled step-by-step based on iterative user-LLMs interactions.

3.3 Efficacy of LLMs in automated error identification and troubleshooting

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE BCVTB-variables SYSTEM "variables.dtd">
<BCVTB-variables>
<!-- ===== OUTPUTS from EnergyPlus (to BCVTB/Ptolemy II) ===== -->
<!-- Site and Zone Environmental Variables -->
<variable source="EnergyPlus">
<EnergyPlus name="ENVIRONMENT" type="OUTDOOR DRY BULB"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="ENVIRONMENT" type="WIND SPEED"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="ENVIRONMENT" type="WIND DIRECTION"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="ZONE1" type="ZONE MEAN RADIANT TEMPERATURE"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="ZONE1" type="ZONE MEAN AIR TEMPERATURE"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="ZONE1" type="ZONE AIR RELATIVE HUMIDITY"/>
</variable>
<!-- EMS/Erl Variables -->
<variable source="EnergyPlus">
<EnergyPlus name="EMS" type="PTACControlStatus"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="EMS" type="PTACSingleCoolingSetpoints"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="EMS" type="WindowOpenAreaFraction"/>
</variable>
<!-- Surface Inside Face Temperatures -->
<variable source="EnergyPlus">
<EnergyPlus name="Zone 1 Roof" type="SURFACE INSIDE FACE TEMPERATURE"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="Zone 1 Floor" type="SURFACE INSIDE FACE TEMPERATURE"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="Zone 1 North Wall" type="SURFACE INSIDE FACE TEMPERATURE"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="Zone 1 South Wall" type="SURFACE INSIDE FACE TEMPERATURE"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="Zone 1 East Wall" type="SURFACE INSIDE FACE TEMPERATURE"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="Zone 1 West Wall" type="SURFACE INSIDE FACE TEMPERATURE"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="Zone 1 Roof Skylight 1" type="SURFACE INSIDE FACE TEMPERATURE"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="Zone 1 SouthWall SouthWindow" type="SURFACE INSIDE FACE TEMPERATURE"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="Zone 1 WestWall WestWindow" type="SURFACE INSIDE FACE TEMPERATURE"/>
</variable>
<!-- PTAC Cooling Energy Variables -->
<variable source="EnergyPlus">
<EnergyPlus name="ZONE1 PTAC" type="TOTAL COOLING ENERGY"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="ZONE1 PTAC" type="SENSIBLE COOLING ENERGY"/>
</variable>
<variable source="EnergyPlus">
<EnergyPlus name="ZONE1 PTAC" type="LATENT COOLING ENERGY"/>
</variable>
<!-- ===== INPUTS to EnergyPlus (from Ptolemy II/BCVTB) ===== -->
<variable source="Ptolemy">
<EnergyPlus schedule="yHVAC_Schedule"/>
</variable>
<variable source="Ptolemy">
<EnergyPlus schedule="tHVAC_Schedule"/>
</variable>
<variable source="Ptolemy">
<EnergyPlus schedule="oWindow_Schedule"/>
</variable>
</BCVTB-variables>
```

Fig. 6 Code generated based on the rule-based external knowledge.

each section of the example MATLAB code, clarify its syntax and external functions, and eventually provided an adaptable template for the co-simulation case. Given the complexity of the OB-TC co-simulation, after defining

```
% BES CFD Co-Simulation Controller Template in MATLAB, this template integrates EnergyPlus (BES), Fluent
(CFD), and MATLAB through BCVTB using BSD sockets. The control signals for HVAC window are determined via
PMV@PMV analysis.

1. Initialize Model Variables
defTim = 900;
simTimWt = 0;
Tset = 0;
TRset = 0;
yHVAC = 0;
oWindow = 0;
v = [yHVAC, tHVAC, oWindow];

2. Initialize Flags (used for socket exchange status)
reVal = 0;
flaRea = 0;

3. Add Path to BCVTB MATLAB Libraries
addpath(genpath('BCVTB_HOME'), 'lib\matlab');

4. Establish the Socket Connection
sockfd = establishClientSocket('socket.cfd');
if sockfd < 0
error('Socket connection to BCVTB failed.');
```

Fig. 7 Code generated based on the code-based external knowledge.

As a Debugger, the LLM demonstrated moderate to high capability in identifying and resolving common errors. For instance, syntax errors and missing variable

definitions were corrected after one or two iterations. Logical errors involving mismatched variable types or sequencing were more challenging but manageable with user feedback. While GPT-4o could not always independently resolve runtime errors involving deep integration (e.g., mismatched variable and type names across tools), its suggestions helped pinpoint the issue and propose a near-complete correction. This highlights that while LLMs can autonomously fix simple or surface-level issues, user-LLMs collaboration remains valuable for complex co-simulation debugging.

3.4 Advantages and limitations

The use of LLMs for OB-TC co-simulation presents several notable strengths. Firstly, LLMs' strong learning adaptability was observed in the case study. GPT can effectively internalize domain rules and code logics from representative examples, enabling itself to imitate them across various coding tasks from simple to complex. Secondly, excellent code generation ability within the building co-simulation domain was witnessed. Most components were correctly produced with clear structure and logics, even for some complex tasks, transfer learning ability from rule/code-based demos was shown by iterative user-LLMs interactions. Lastly, the LLMs demonstrated reliable self-troubleshooting capability in most cases, revising wrong codes based on runtime errors and feedback.

Despite these strengths, several limitations were observed. Firstly, GPT occasionally produced verbose or unnecessary code segments, especially when prompts lacked precision. Secondly, the high computational cost remains a limitation of existing GPT models, as extended interaction loops and iterative prompting can be resource-intensive, particularly for large-scale simulations. Thirdly, the model occasionally returned inconsistent or contradictory answers to similar prompts, affecting reliability. Lastly, lack of fine-tuning is another important limitation, without domain-specific fine-tuning, the LLM could not always produce expert-level outputs for tasks requiring deep knowledge of simulation software, future work requires more focus on the domain-tuned LLMs and structured prompt templates to improve performance in specialized engineering applications.

4. CONCLUSIONS

This study explored the novel application of large language models, specifically GPT-4o, in automating the co-simulation of occupant behavior and thermal comfort. Through a multi-role framework, comprising

Analyst, Coder, and Debugger functions, we evaluated the effectiveness of LLMs in structuring co-simulation workflows, generating code, and performing iterative debugging. The conclusions can be drawn as follows:

(1) LLMs can handle various OB-TC co-simulation tasks, such as establishing co-simulation frameworks, generating platform-specific code, and correcting execution errors with minimal human intervention.

(2) For routine or moderately complex tasks, LLMs can reduce the time and effort typically required by manual coding, thereby lowering the entry barrier for co-simulation applications. The quality of outputs strongly depends on the clarity, structure, and completeness of prompts. Well-formulated examples and external references significantly enhance LLM performance.

(3) While LLMs show promise in many aspects, expert oversight remains essential, particularly for tasks involving high integration complexity or domain-specific edge cases.

ACKNOWLEDGEMENT

The work was supported in part by the Research Grants Council of Hong Kong SAR under Grant 14200524, and in part by the InnoHK initiative of the Innovation and Technology Commission of the Hong Kong Special Administrative Region Government via the Hong Kong Centre for Logistics Robotics as well as CUHK Direct Grant for Research No. 4055230.

REFERENCE

- [1] Zhao, W.X., K. Zhou, J. Li, et al., A survey of large language models. arXiv preprint arXiv:2303.18223, 2023.1(2).
- [2] Liu, M., L. Zhang, J. Chen, W. A. Chen, Z. Yang, J. Lo, J. Wen, and Z. O'Neill, Large language models for building energy applications: Opportunities and challenges. Building Simulation, 2025.
- [3] Zhang, L. and Z. Chen, Opportunities of applying Large Language Models in building energy sector. Renewable and Sustainable Energy Reviews, 2025. 214: p. 115558.
- [4] Zhang, L., Z. Chen, and V. Ford, Advancing building energy modeling with large language models: Exploration and case studies. Energy and Buildings, 2024. 323: p. 114788.
- [5] Li, Y., L. Li, X. Cui, and P. Shen, Coupled building simulation and CFD for real-time window and HVAC control in sports space. Journal of Building Engineering, 2024. 97: p. 110731.