ChatDC: Geometric-aware Data Center Digital Twin Generation via Large Language Models

MINGHAO LI, Nanyang Technological University, Singapore, Singapore
RUIHANG WANG, Nanyang Technological University, Singapore, Singapore
XIN ZHOU, Jiangxi Science and Technology Normal University, Nanchang, China
ZHAOMENG ZHU, Nanyang Technological University, Singapore, Singapore
YONGGANG WEN, Nanyang Technological University, Singapore, Singapore
RUI TAN, Nanyang Technological University, Singapore, Singapore
HUIWEN ZHENG, Global Data Solutions Limited, Shanghai, China
STUART KENNEDY, DayOne Data Centres Limited, Singapore, Singapore

A modern data center is supported by an Internet of Assets (IoA), a specialized Internet of Things (IoT), where the sim-ready assets encompass physical properties and connected data streams for passive data collection and proactive simulation. A digital twin (DT) is a virtual replica of the IoA system with a proper level of abstraction, integrating asset-level dynamics models to simulate system-wide behavior, prototype designs, and conduct what-if analysis. However, creating high-fidelity DTs is hindered by the manual effort needed to encode complex geometric layouts and domain-specific design constraints. While large language models (LLMs) offer automation potential, existing methods struggle to generate geometrically plausible and functionally valid DT scenes due to limited domain integration. To bridge the gaps, we propose ChatDC, a conversational system that leverages LLMs to automate data center digital twin generation through a Segment-Generate-Optimize (SGO) workflow. ChatDC integrates domain knowledge via a dedicated code library named DCBuild and employs SGO to decompose user prompts, generate initial structures, and optimize layouts in compliance with data center design constraints. Evaluation shows that ChatDC outperforms other baselines with 98% success rate on scratch generation tasks and reduces the average makespan by 10× . Ablation study reveals that the SGO design increases the generation success rate by 65% at most. Furthermore, computational fluid dynamics simulations validate the physical plausibility, confirming the readiness of generated DTs for real-world analysis.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; • **Applied computing** → *Industry and manufacturing*; • **Computer systems organization** → **Embedded and cyber-physical systems;**

Additional Key Words and Phrases: Internet-of-things, digital twin, large language model, data center

This work was supported by the National Natural Science Foundation of China (No. 62262026), the Jiangxi Natural Science Foundation (No. 20232BAB202020), and in part by Digitalland Innovation and Technology Pte Ltd.
Authors' Contact Information: Minghao Li (corresponding author), Nanyang Technological University, Singapore, Singapore; e-mail: minghao002@e.ntu.edu.sg; Ruihang Wang, Nanyang Technological University, Singapore, Singapore; e-mail: ruihang001@e.ntu.edu.sg; Xin Zhou, Jiangxi Science and Technology Normal University, Nanchang, Jiangxi, China; e-mail: zhouxin@jxstnu.edu.cn; Zhaomeng Zhu, Nanyang Technological University, Singapore, Singapore; e-mail: zhaomeng.zhu@ntu.edu.sg; Yonggang Wen, Nanyang Technological University, Singapore, Singapore; e-mail: ygwen@ntu.edu.sg; Rui Tan, Nanyang Technological University, Singapore, Singapore; e-mail: tanrui@ntu.edu.sg; Huiwen Zheng, Global Data Solutions Limited, Shanghai, China; e-mail: huiwen.zheng@gds-services.com; Stuart Kennedy, DayOne Data Centres Limited, Singapore, Singapore; e-mail: stuart.kennedy@dayonedc.com.

ACM 2577-6207/2026/04-ART17
https://doi.org/10.1145/3772078

ACM Reference Format:
Minghao Li, Ruihang Wang, Xin Zhou, Zhaomeng Zhu, Yonggang Wen, Rui Tan, Huiwen Zheng, and Stuart Kennedy. 2026. ChatDC: Geometric-aware Data Center Digital Twin Generation via Large Language Models. *ACM Trans. Internet Things* 7, 2, Article 17 (April 2026), 26 pages. https://doi.org/10.1145/3772078

1 Introduction

In modern data centers, assets such as servers, **air conditioning units** (**ACUs**), and smart racks are computerized and networked physical objects equipped with sensors for self-awareness and self-regulation [46]. Servers monitor their operational status, power consumption, and thermal conditions, while ACUs dynamically adjust cooling based on temperature setpoints, and smart racks measure and transmit power usage data. This integration of sensing, computation, and networking forms as an ***Internet of Assets*** (**IoA**), a specialized local-area **Internet of Things** (**IoT**) [6], where the sim-ready assets encompass accurate physical properties, behavior, and connected data streams [30]. Besides traditional IoT frameworks focused on passive data collection, IoA enables bidirectional virtual-physical interactions: each asset maintains a physics-based dynamics model, which supports high-fidelity simulations to predict system behavior and preemptively mitigate risks. For instance, when deploying **high-performance computing** (**HPC**) infrastructure, an IoA system can simulate how server-level power and heat fluctuations propagate to alter cold aisle temperatures, allowing operators to adjust cooling strategies proactively. A **digital twin** (**DT**) integrates these asset-level dynamics models as a virtual replica of the large-scale data hall system for holistic simulations, enabling operators to simulate system behavior, validate prototype design, and conduct what-if analysis before physical deployment [45].

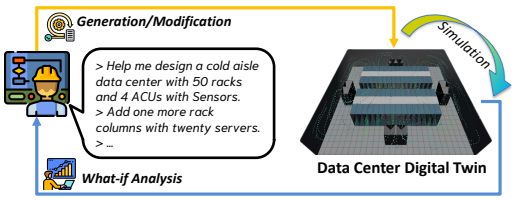

Fig. 1. ChatDC generates data center DTs with prompts. The generated DTs further assist operators to perform what-if analysis.

DTs are created during the system design phase with offline specifications and updated as the operation progresses. These models allow users to prototype the physical system design and conduct various what-if analyses before actual deployment. DTs have been used in many industries, including manufacturing [19], smart grids [13], and data centers [23]. In particular, data center DTs are advantageous in facilitating reliable design and proactive operations. For example, operators can evaluate potential energy-efficient cooling control policies in the digital environment [35, 36]. Thus, the evaluation will not risk the physical system. To describe the physical properties of the built-in facilities, existing tools often use standard and structural data, such as JSON and XML, for configuration [5]. However, manually configuring a data center DT often requires domain expertise and intensive efforts as the system scale increases. For instance, a large-scale data center can host thousands of IT devices and tens of ACUs [46] that need to be properly positioned for effective cooling. Editing the configuration of such a large system is time-consuming and subjected to human errors, such as typos, incorrect data representation, and flawed system design. Improper designs during the system prototyping stage may result in space waste or even raise thermal safety concerns [21].

Conventional automated DT generation methods usually adopt the no-code platform [5], the programming-based editor [29], and the scanning techniques [1] to populate the virtual objects based on predetermined logic. However, applying these tools still requires human efforts and expertise to fulfill the design objectives [3]. **Large language models** (**LLMs**) are good candidates

for understanding the design intentions and facilitating DT generation due to their human-level comprehension and reasoning abilities [50]. LLMs have shown remarkable ability in automated 3D scene generation. Current studies have explored LLM-based code generation in 3D programming engines (Blender, Unity, Omniverse, etc.) to automate object creation. The 3D-GPT [43] achieves instruction-driven 3D modeling by leveraging LLMs to collaborate task dispatch, conceptualization, and modeling agents in Blender. Similarly, the LLMR [10] utilizes LLMs for real-time creation and modification of interactive mixed reality experiences empowered by Unity. Different from human-centric indoor scenes, data centers host numerous facilities that need to be properly positioned with domain-specific rules. For example, the supply of cold air is commonly pumped into the room below the raised floor and then allowed to rise through perforated tiles for effective cooling [38]. The facility placement can affect the data hall airflow and thermal distributions, which are critical for resilient operations. However, the above LLM-based approaches do not incorporate specific design requirements in the DT generation.

To bridge the gaps, we adapt the LLM-based approaches to complex data center DT generation with optimized design and propose *ChatDC*. Through descriptive prompts, ChatDC facilitates automated data center DT generation for various what-if analyses as shown in Figure 1. To tackle the limitations of LLMs in geometric plausibility and domain-specific reasoning, ChatDC is designed with two parts: (i) a code library for domain specialization, and (ii) an LLM orchestrator for geometric plausibility. We first integrate the domain knowledge into a code library named *DCBuild*, which internalizes data center geometric design principles to enable executable implementations of data center DTs. To enhance the geometric plausibility of DT generation with **natural language** (**NL**) instructive prompts, we introduce the **Segment-Generate-Optimize** (**SGO**) framework. Specifically, the SGO, acting as an orchestration of LLMs, aims to enhance generation efficiency and accuracy concerning both configuration format and 3D geometric optimality. Specifically, it first breaks down the tasks into subtasks for generation and optimization purposes. With the DCBuild library documentation, the SGO then generates and optimizes DTs. ChatDC provides executable and actionable DTs for simulation and optimization for data centers with LLM-assisted generation.

We evaluate ChatDC on various DT generation tasks that cover generation from scratch and modification from existing models. The experimental results reveal its outstanding performance, scoring a 98% generation success rate and reducing average makespan by 10× , which outperforms commonly used prompting methods and our previously proposed framework, ChatTwin [22]. Additionally, we perform an ablation study on the SGO framework. Our findings highlight the necessity of the designed modules within the framework, as the design boosts the generation success rate by 65%. Meanwhile, we evaluate the generation efficiency across varying scales of data centers and different levels of prompt complexity. These findings illustrate the balance between the makespan and the success rate within the SGO framework, while simultaneously emphasizing ChatDC's capability to understand instructive prompts for constructing data centers. Moreover, we execute **computational fluid dynamics** (**CFD**) simulations on various representative DTs and adopt quantitative metrics for evaluating the outcomes. These assessments confirm the validity of the executable DTs generated by ChatDC, further offering valuable insights for operators to prototype the designs in the pre-construction phase of data centers.

Our main contributions are summarized as follows:

(1) We present ChatDC with the SGO to tackle complex design problems in the data center industry using LLMs.
(2) We conduct extensive evaluations to test the performance, including the success rate, incremental generation, and efficiency, demonstrating the practical DT validity using CFD simulation and quantitative analysis.

(3) We showcase how LLMs extend beyond human-machine interaction, underscoring their broad potential in industrial applications, particularly in DT-based data center design and construction.

The rest of the article is organized as follows: Section 2 presents prior work and methods. Section 3 presents the preliminary knowledge of the data center, the geometric design principles, and our system design considerations. Section 4 provides an overview of ChatDC, domain-knowledge embodied DCBuild, and the details of each module in the SGO. Section 5 presents several exemplar applications. Section 6 includes a comprehensive performance evaluation, an ablation study, and a validation for the quality of ChatDC's output DTs. Section 7 discusses several related issues. Section 8 concludes this article.

2 Related Work

This section introduces the studies related to generative 3D scene creation, design automation for building system, and data center intelligent design.

2.1 Generative 3D Scene Creation

Research on generative 3D objects has gained popularity in recent years. Chang et al. [7] show a text-to-3D technology, which extracts explicit constraints on the objects and augments the constraints with learned spatial knowledge to generate the whole scene. Ma et al. [28] utilize NL for generative 3D scene creation, transforming user commands into semantic scene graphs, and synthesizing new 3D scenes from large databases. With the appearance of ChatGPT, interactive 3D scene generation has been a significant hotspot in the LLM research field. 3D-GPT [43] builds a bridge between LLMs and Blender to generate a large natural scene by instructive prompts. The LLMR [10] framework utilizes LLMs for real-time creation and modification of interactive mixed reality experiences. While previous studies have demonstrated the LLMs capabilities in 3D modeling, they are not dedicated to generating 3D scenes for data center DTs, which have specific design requirements and higher geometric complexity.

2.2 Design Automation for Building Systems

Automated design for building systems has been studied and applied to different building types. Back in 1997, Papamichael et al. [33] provided a software environment that allows multiple visualization and analysis tools and, therefore, simplifies the navigation throughout the building design process. Jia et al. [18] present a methodology that involves abstracting and formalizing components of smart buildings from high-level specifications and mapping them to physical implementation. Liu et al. [26] propose a solution that combines computer-processable layout design rules and some rule-based design algorithms with mathematical algorithms to automate the design. While previous studies show various automated design methods for buildings, they do not incorporate language-based design intentions to automate the data center building construction, which would significantly reduce manual effort and improve efficiency.

2.3 LLM Agents for Tool Use

Recent studies on LLM agents for tool use have demonstrated promising results in open-domain tasks [25]. Early research begins by utilizing LLMs to generate executable code as intermediate reasoning steps [14]. This method offloads the computational aspects of problem-solving to the interpreter, like solving math problems through generated Python code. To improve the stability, LLM vendors design interfaces for LLM tool use. OpenAI's function call feature enables LLMs to autonomously select and invoke external functions within their responses [31]. Similarly, **model context protocol** (**MCP**) provides a standardized communication protocol that allows LLMs

Table 1. Summary of Notations

Symbol	Definition	Symbol	Definition
x_w, x_l	width and length of the data hall	$\mathcal{T}_g$, $\mathcal{T}_o$	set of the subtasks for DT generation and optimization
U	space utilization of the data hall	V	set of facilities in the data center
w_{aisle}, l_{aisle}	width and length of the aisle	G	hierarchical representation of the data center DT
ξ_{aisle}, ξ_{acu}	gap between each rack column and gap between each ACU	$\mathfrak{L}$	pretrained LLM
w_{rack}, l_{rack}	width and length of the rack	P_s	prompt templates for segmenting subtasks
w_{acu}, l_{acu}	width and length of the ACU	P_ϕ	prompt templates for generating codes of DCBuild
n_{rack}, n_{acu}	number of racks and ACUs in the data hall	R	set of optimization options for the imperfect DT
γ, μ, ν	layout, padding, margin of the data hall	$\mathcal{L}$	set of DCBuild function descriptions
$\mathcal{I}$	NL input for DT generation	$\mathcal{J}$	set of DT description file fragments

to interact with external data sources and tools [4], like retrieving real-time weather information via API calls. With the increasing complexity of the tasks, LLM agent systems [37, 41, 49] integrate planning, multi-step orchestration, and tool use to autonomously solve complex workflows by combining programmatic reasoning and API access, like web browsing, data analysis, and content creation. While these systems demonstrate strong performance in open-domain tasks, they have not been designed for complex IoT scenarios. DT construction requires physics-plausible spatial layouts that simple function call interfaces and generic agent pipelines cannot achieve.

3 Data Center Overview and Design Requirement

This section starts with preliminary knowledge of data centers. Then, we introduce several geometric design principles and an optimization problem in data centers. Finally, we list the design considerations of our approach. Table 1 summarizes the notations used in this article.

3.1 Data Center Preliminary

A data center is a complex IoA system that occupies extensive space and hosts numerous cooling facilities and IT equipment. To ensure the operation safety and effective cooling of the IT equipment, the ACUs supply cold air and absorb the heat generated from the IT outlet with the specific containment design. Since the geometry feature essentially affects the data center's safety, efficiency, and sustainability, the hosted facilities should adhere to certain positioning rules when included in a DT to prototype the data center design.

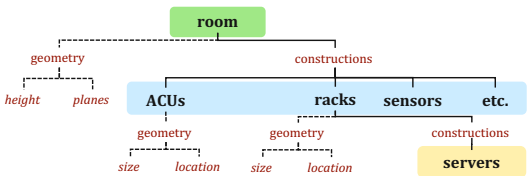

Fig. 2. The hierarchical structure of DT for modeling data centers. The first level refers to the "Room". The second level includes "Racks", "ACUs", "Sensors", and other facilities within the "Room". The third level contains only the "Servers" in the data center context.

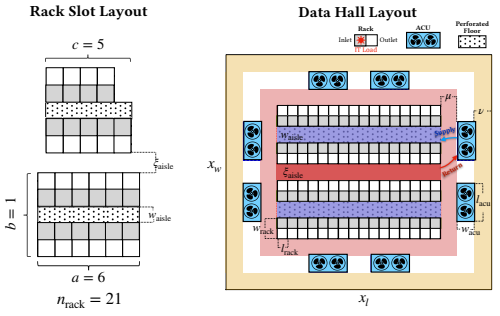

Fig. 3. (left) The designed layout for racks and aisles in rows and pair columns. (right) The recommended layout for an example data center.

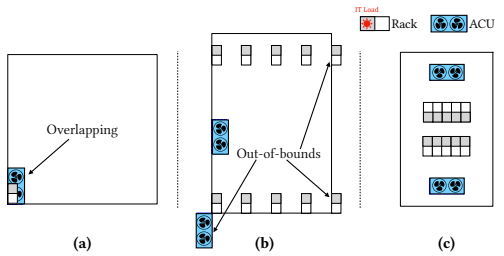

Fig. 4. Comparisons among three scene description documents: (a) a fault DT generated by GPT-4 without in-prompt layout design principles, (b) an imperfect DT generated and optimized by GPT-4 with in-prompt layout design principles, and (c) the ideal data center layout.

Similar to existing DT systems, the DT of a data center is defined in a hierarchical structure. The structure of the data center DT is illustrated in Figure 2, which utilizes three levels to represent all facilities in the data center. Specifically, the top level defines only one item, i.e., the "`Room`" object, which refers to the data hall. The "`Room`" has several children, i.e., "`Racks`", "`ACUs`", "`Sensors`", and so on, which is defined at the second level. In this article, we focus on the locations of racks and ACUs for optimization. The third level defines the "`Servers`" object that is accommodated under the "`Racks`". We also model the containment design with "`Openings`" and "`Boxes`" in the second level. This tree architecture exhibits the hierarchical nature inherent in the structure of a data center.

3.2 Data Center Layout Generation Principles

We formulate the data center layout generation problems concerning room utilization and cooling requirements. Eventually, we expect the generated DT to have an optimal geometry layout that complies with IT equipment cooling requirements. The decision variables used to optimize the data center utilization include the number of racks in the column, the number of aisle rows, and the number of racks in the extra pair column.

3.2.1 Room Utilization. We consider a typical rectangular design of the data hall with width and length denoted by x_{w} and x_{l}, respectively. An optimal data hall design should optimize the space utilization defined as

$$U = \frac{\sum_i^k s_i}{x_{\mathrm{w}} \times x_{\mathrm{l}}}, \tag{1}$$

where k is the total number of facilities and the bottom area of each hosted facility is denoted as s_i. Let n_{rack} and n_{acu} denote the total number of racks and ACUs, respectively. Since each facility has three dimensions to determine its location, the incremental number of facilities imposes high complexity for solving the optimization problem. Thus, the solution space of this optimization problem is very large, considering the locations of every rack and ACU as the variables. Due to the NP-completeness [8], we plan to solve the problem via a heuristic way by the following rules.

Rule 1: We set the racks in pair columns with the unified aisle width w_{aisle}. The gap between each pair column is denoted as ξ_{aisle}. We use slots to determine the positions of the racks. The rack

slot layout is shown as the left part of Figure 3. Thus, we can set the layout of the rack slots by

$$n_{\text{rack}} = \begin{cases} 2(a \times b + c), & n_{\text{rack}} \text{ is even}, \\ 2(a \times b + c) - 1, & \text{else}. \end{cases} \tag{2}$$

where a is the number of racks in columns, b is the number of aisle rows, and c is the number of racks in the extra pair column. Since a, b, and c decide the layout of the racks, they are regarded as the variables of the optimization problem. We denote the variable layout as $\gamma = (a, b, c)$ and Equation (2) as the equality constraint.

Rule 2: Racks should be placed in the central area as Figure 3, while ACUs should be evenly arranged around the group of racks with gap ξ_{acu}. The margin μ of the area of racks and the padding ν of the data hall should be adjusted to satisfy the cooling requirements. This data hall layout design rule is shown as the right part of Figure 3. The data hall width x_{w} can be calculated by the rack's width w_{rack}, length l_{rack}, the ACU's width w_{acu} and rack layout γ as

$$x_{\text{w}} = \begin{cases} a \times w_{\text{rack}} + 2(\mu + \nu + w_{\text{acu}}), & a \geq c, \\ c \times w_{\text{rack}} + 2(\mu + \nu + w_{\text{acu}}), & \text{else}. \end{cases} \tag{3}$$

Then, the data hall length x_{l} is derived by the aisle's width w_{aisle}, the gap ξ_{aisle}, the rack's length l_{rack}, the ACU's width w_{acu}, and the rack layout γ with two parameters μ and ν as

$$x_{\text{l}} = 2(b + 1) \times l_{\text{rack}} + (b + 1) \times w_{\text{aisle}} + b \times \xi_{\text{aisle}} + 2(\mu + \nu + w_{\text{acu}}). \tag{4}$$

Then we regard Equations (3) and (4) as the equality constraints.

Rule 3: We ensure there is no facility placed outside the hall. The hall is rectangular without any extra pillars in the room. Considering the number of ACUs, denoted by n_{acu}, we have the constraints of room width x_{w} and length x_{l}, respectively, as

$$\begin{aligned} x_{\text{w}} &\geq m \times l_{\text{acu}} + (m - 1) \times \xi_{\text{acu}}, \\ x_{\text{l}} &\geq m \times l_{\text{acu}} + (m - 1) \times \xi_{\text{acu}}, \end{aligned} \tag{5}$$

where $m = \lceil \frac{n_{\text{acu}}}{4} \rceil$ means the largest possible number of ACUs on one side. Based on the three rules, we formulate an optimization problem of integer programming:

$$\begin{aligned} &\underset{a,b,c \in \mathbb{N}}{\text{maximize}}\ U \\ &\text{subject to Equations (2), (3), (4), (5).} \end{aligned} \tag{6}$$

While the original problem exhibits a vast solution space, we reduce its complexity by leveraging domain-specific constraints, effectively narrowing the optimization to three key parameters. This dimensionality reduction allows us to employ grid search to explore the restricted solution space. In other words, the time complexity of the algorithm is $O(n^3)$. The thermal effects are influenced by the hyperparameters, including the aisle's width w_{aisle}, the rack's gap ξ_{aisle}, the ACUs' gaps ξ_{acu}, the margin μ, and the padding ν.

3.2.2 Facility Placement. Beyond the geometric correctness of facility layout, data hall design also needs to incorporate domain knowledge for facility requirements. To achieve this, we focus on containments (passageway), ACUs (cooling source), and servers (heat source) with several design principles based on prior data center research [34, 39].

Aisle Containment Design. Data center aisle isolation is a common principle used in data centers to maintain temperature setpoints and lower energy costs. The main objective of this approach is to maximize cooling efficiency by minimizing the mixing of hot and cold air. Let $L_{\text{ctm},i}$ denote the containments with locations, and its size with length $l_{\text{ctm},i}$, width $w_{\text{ctm},i}$, and height $h_{\text{ctm},i}$, where $i \in N^+, i \leq b + 1$. The containment is installed as the blue rectangle area in Figure 3.

We consider two primary types of aisle containment designs with different containment heights. For *hot aisle containment* design, the hot exhaust air from the servers is contained and directed back to the ACU return via the contained hot aisles, while the cold air is spread in an uncontained area. Similarly, the cold intake air for the servers is contained and directed to the server inlets via *cold aisle containment*, while the hot air is spread in an uncontained area.

ACU Orientation. In a data center, ACUs should be oriented to supply cold air into cold aisles and absorb hot air from hot aisles, facilitating proper airflow. Two ACU types are set for data centers with and without raised floors. For centers without raised floors, ACU outlets face the room center with an air inlet at the top. In rooms with raised floors, ACUs have a top opening for hot air absorption and a bottom outlet to send cool air through the gap between the floor and the ground.

Server Slot Placement. For rack-mounted servers, managing heat dissipation is critical, which can be regulated by the gaps between servers, denoted as ξ_{server}. The slot placement for each server in a rack holding n_{server} servers is defined as $s_i = \sum_{k=1}^{i-1} o_k + \xi_{\text{server}}$, where o_i is the server's slot occupation. Blanking panels are used to prevent air recirculation, and servers are placed from the bottom up to ensure rack stability.

3.3 ChatDC Design Considerations

While LLMs have exhibited spatial reasoning capability in layout design [11], they are not omnipotent enough to generate a data center DT in accordance with specific design principles. Figure 4 shows two typical flaws generated by GPT-4, with or without in-prompt layout design principles. From the results, the generated facilities are out of the data hall boundary even with specifically engineered prompts. These poor layouts disqualify the DT's usage for effective data center design during the pre-construction phase. There are two major barriers for current LLMs to generate plausible data center DTs.

3.3.1 Lack of Domain-specific Knowledge and Constraints. LLMs are trained on general-purpose datasets and lack the domain-specific knowledge required for specialized tasks. This limitation is exacerbated by their inability to incorporate domain-specific constraints effectively. LLMs struggle to adapt to domain-specific tasks due to the heterogeneity of domain data, the sophistication of domain knowledge, and the uniqueness of domain objectives. For example, in medical diagnosis or legal reasoning, LLMs often generate plausible-sounding but incorrect answers due to their lack of specialized knowledge [24]. Besides, current LLMs are not inherently designed to incorporate domain-specific constraints, such as energy efficiency, structural integrity, or cooling requirements in data center design [47]. Thus, teaching LLMs to learn domain-specific knowledge and constraints is critical in the framework design.

3.3.2 Limitations in Geometric Plausibility. LLMs, despite their impressive text generation capabilities, struggle with tasks requiring spatial reasoning and geometric plausibility [44]. These limitations are not model-specific but are inherent to the architecture and training paradigms of current LLMs. Hybrid approaches are needed to ensure geometric plausibility, which LLMs alone cannot achieve. Besides, LLMs have limited performance when solving arithmetic reasoning tasks and often provide incorrect answers. As LLMs excel at pattern recognition in discrete tokens rather than precise mathematical computation, even the current SOTA approach of mathematical reasoning still fails to provide stable outputs on some simple problems[40]. This fragility further underscores their lack of deep, adaptable understanding of spatial and logical principles. Thus, adopting LLMs to generate plausible data center DTs requires additional frameworks to handle geometric constraints effectively.

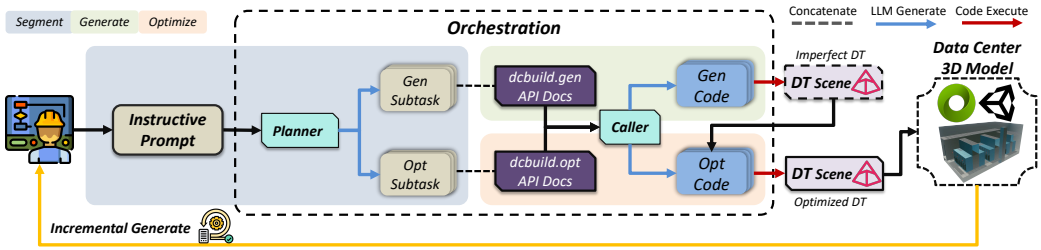

Fig. 5. *The overview of automating DT generation and optimized design via ChatDC.* ChatDC utilizes LLMs as the Planner for segmentation and the Caller for generation and optimization. First, the Planner decomposes the dialog input into several subtasks for generation and optimization. Then, each generation subtask is gathered with its corresponding documentation and sent to the Caller. Through the codes generated by the Caller, ChatDC first creates an imperfect DT scene. Then, the imperfect DT scene is used in the optimization phase and gradually modified by the optimization codes. Finally, the optimized DT scene object can be visualized by 3D engines with CFD simulation results for the user to upgrade the design of the data center.

4 ChatDC Approach Design

In this section, we first show the overview of the ChatDC approach. ChatDC consists of two parts, (i) a code library for domain specialization, and (ii) an LLM orchestrator for geometric plausibility. Then, we introduce how we inject the code library. Finally, we elucidate the orchestration design and how each module collaborates effectively within the system.

4.1 Approach Overview

LLMs are capable of semantic planning [42] and code generation [14] with instructive prompts given by users. However, generating DTs for multi-layered data center scenarios is challenging due to the complexity of the structural 3D industrial environment. Existing LLM function call capabilities only offer reliable tool access but lack the hierarchical planning and spatial reasoning capabilities for DT generation. Generic LLM agents are capable of reasoning for task decomposition, but cannot infer the strict geometric relation between each facility. Promoting these features of DTs in optimized layouts requires a framework that interprets user intentions, generates high-quality code, and understands domain knowledge for optimal design.

Previously, we proposed ChatTwin [22] to tackle the description file generation by the Segment-and-Generate workflow and the heuristic optimization strategy. To achieve better generation performance and more physics-plausible data center design, we propose ChatDC, an LLM-empowered geometric-aware DT generation framework for data centers. Instead of directly generating the scene description file [22], ChatDC leverages the **program-aided language** (**PAL**) [14] technique to generate the DT objects based on a code library. The library's programmatic approach enables ChatDC for scalable DT generation by two steps: (i) LLMs orchestrate high-level design logic, while (ii) DCBuild rigorously enforces constraints through deterministic code execution. This synergy ensures the generated DTs are both functionally valid and aligned with DC industry best practices. The implementation of our SGO workflow is shown as Figure 5.

4.2 DCBuild: A Code Library for Data Center Domain Knowledge and Constraints

We present DCBuild, a Python library designed to codify data center domain expertise and automate geometric-aware data center design constraints. DCBuild encapsulates industry-standard practices, spatial-configuration rules, and facility interdependencies into programmable modules, enabling systematic generation and optimization of DT scenes. DCBuild covers assets mentioned

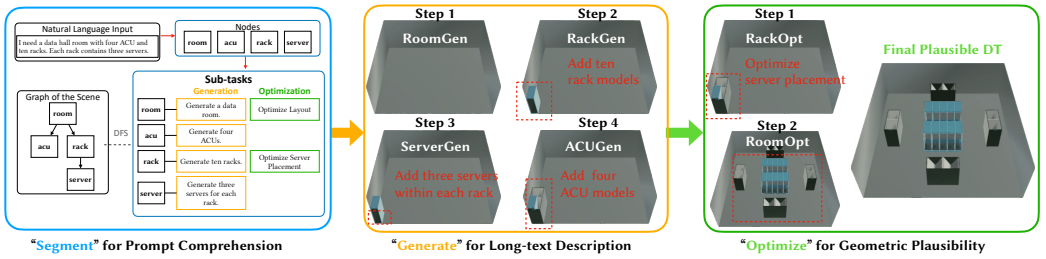

Fig. 6. The internal working of SGO workflow in steps. (1) **Segmentaion**: We process the NL task to graph-based sub-tasks. As the example prompt mentions four facilities, the initial prompt is segmented into four sub-tasks, related to the generation and optimization of Room, Rack, Server, and ACU. (2) **Generation**: By the instructions of each sub-task for generation, it leverages DCBuild to gradually synthesize an imperfect DT with the four components in DFS order. These components are assets that are executable-ready for simulation. (3) **Optimization**: Finally, the sub-tasks for optimization of room and rack guide the generated DT scene step by step to be a geometric plausible DT.

in Section 3.1 and layout design principles as shown in Section 3.2. The library is structured into two core modules, *Generator* and *Optimizer*.

Generator Module. This module translates high-level design intents into compliant DT fragments through domain knowledge-guided code generation. It first comprehends domain terminology in the input prompt and further converts NL information into code snippets of predefined sim-ready assets (e.g., rack, server, and ACU). For example, the RackGenerator class encodes geometric attributes and power specifications to produce rack models, while the ACUGenerator incorporates parameterized thermodynamic configurations to describe the physical attributes of cooling units. By programmatically assembling these fragments, DCBuild synthesizes an initial DT scene that adheres to foundational data center standards.

Optimizer Module. This module refines raw DT scenes through constraint-driven adjustments. Classes like RackOptimizer and RoomOptimizer apply rule-based corrections (e.g., aligning rack orientations for hot/cold aisle containment, and balancing power circuits) and support manual overrides. The optimizer evaluates facility placements against domain-specific constraints, such as safety margins and thermal efficiency thresholds, and iteratively adjusts geometric attributes or installs/removes facilities to ensure compliance.

By integrating domain knowledge directly into code, DCBuild bridges semantic planning via LLMs and precise engineering requirements in the DC industry. For instance, rack and cooling unit placements are optimized using thermodynamic knowledge rather than heuristic prompts, ensuring physically viable layouts.

4.3 SGO: An LLM Orchestrator for Plausible Scene Generation

To bridge the gap between human input in NL and executable code for DT generation, it is vital to let LLMs comprehend prompts and generate relevant codes. We thereby propose a new SGO workflow as an orchestration to tackle both automated generation tasks and optimized design of data center DTs with DCBuild.

4.3.1 Segment. Instead of generating all letters of a description file at once, the SGO first cuts the NL input $\mathcal{J}$ into several parts denoted as $\mathcal{T}_g = \{t_{g,1}, t_{g,2}, t_{g,3}, \ldots, t_{g,i}, \ldots, t_{g,n}\}$, where $\mathcal{T}_g$ is the set of all subtasks by users and $t_{g,i}$ represents each subtask. As illustrated in Figure 6, an NL input to generate a data center scene like "*I need a data hall room with four ACU and ten racks. Each rack contains three servers.*". The task will be further decomposed into multiple generation subtasks related to

each node in the scene graph, such as "*Generate a data room*", "*Generate four ACUs*", "*Generate ten racks*", and "*Generate three servers for each rack*". In the scene of the data center DT, the facilities are always in a nested structure. For example, the racks are placed in a room and the servers are installed in a rack slot. We denote the facilities as $V = \{v_1, v_2, v_3, \dots, v_n\}$, where each facility v_i is associated with its parent facility v_p and the directed edge $e_j = (v_p, v_i)$. Their nested tree structure can be predefined by a hierarchical representation $G = \{V, E\}$, where $E = \{e_1, e_2, e_3, \dots, e_j, \dots, e_n\}$. Before decomposing the initial input $\mathcal{I}$ into subtasks, the SGO first identifies all facilities mentioned in the input as nodes by $V = \mathfrak{L}(P_v, \mathcal{I})$, where P_v is the prompt for identifying the nodes and $\mathfrak{L}$ is the pretrained LLM. It appends each hierarchical relation (v_i, v_p) into E for each detected facility v_i, where v_p is the parent node of v_i. Then, $t_{\mathrm{g},i}$ can be extracted from the original input $\mathcal{I}$ by its corresponding node facility v_i and its parent node facility v_p as

$$t_{\mathrm{g},i} = \mathfrak{L}\left(P_{\mathrm{s,g}}(v_i, v_p, \mathcal{I})\right), \tag{7}$$

where $P_{\mathrm{s,g}}$ is the prompt template for segmenting generation subtask.

The set of predefined optimization options for each facility is written as $\mathcal{R} = \{R_1, R_2, R_3, \dots, R_n\}$. Each facility's optimization options list R_i contains k options as defined in $R_i = \{r_{i,1}, r_{i,2}, r_{i,3}, \dots, r_{i,k}\}$. We denote the set of optimization subtasks as $\mathcal{T}_{\mathrm{o}} = \{t_{\mathrm{o},1}, t_{\mathrm{o},2}, t_{\mathrm{o},3}, \dots, t_{\mathrm{o},i}, \dots, t_{\mathrm{o},n}\}$, where $t_{\mathrm{o},i}$ represents the optimization task for facility v_i. As illustrated in Figure 6, the optimization subtasks for room and racks can be "*Optimize layout*" and "*Optimize server placement*". Then, $t_{\mathrm{o},i}$ can be derived from the original input $\mathcal{I}$:

$$t_{\mathrm{o},i} = \mathfrak{L}\left(P_{\mathrm{s,o}}(R_i, \mathcal{I})\right), \tag{8}$$

where R_i is the optimization options list of facility v_i and $P_{\mathrm{s,o}}$ is the prompt for segmenting the optimization task.

The segmented prompts reduce the complexity of the original generation task and are expected to improve the optimization quality. Through the generation process in the SGO, we subsequently utilize prompt engineering and PAL to get the completion codes of each subtask.

4.3.2 Generate. We can get a dictionary of generation subtasks $\mathcal{T}_{\mathrm{g}}$ with each facility as the key and its description as the value from the above task segmentation. The DCBuild class methods are wrapped in utility functions with descriptions of outputs and arguments. We denote the set of all function descriptions as $\mathcal{L}_{\mathrm{g}} = \{L_{\mathrm{g},1}, L_{\mathrm{g},2}, L_{\mathrm{g},3}, \dots, L_{\mathrm{g},i}, \dots, L_{\mathrm{g},n}\}$ for all facilities, where $L_{\mathrm{g},i}$ represents the set of function descriptions related to facility v_i.

Given the generation subtasks $\mathcal{T}_{\mathrm{g}}$ and the API descriptions $\mathcal{L}$, we can get a set of DT file fragments, denoted as $\mathcal{J} = \{j_1, j_2, j_3, \dots, j_i, \dots, j_n\}$, where the fragment j_i for facility v_i is calculated as

$$j_i = \phi\left(\mathfrak{L}(P_{\phi,\mathrm{g}}(t_{\mathrm{g},i}, L_{\mathrm{g},i}))\right), \tag{9}$$

where ϕ is the Python executable kernel and $P_{\phi,\mathrm{g}}$ is the prompt template of the Caller for code generation, respectively. Examples of the generated fragments in $\mathcal{J}$ for a rack and an ACU are provided in Appendix A. The SGO workflow follows the order of **deep-first search** (**DFS**) of G to solve all generation subtasks $\mathcal{T}_{\mathrm{g}}$. Finally, the DT file fragments $\mathcal{J}$ are concatenated into an imperfect DT Ω. As shown in the middle part of Figure 6, this process, guided by the generation sub-tasks, gradually synthesizes data center DT scenes. It begins with room generation to generate an empty room. Then, ten racks are generated in the empty room. Each rack further adds three server models. Finally, four ACUs are added to the room together with racks and servers.

4.3.3 Optimize. Although we can get the DT description file Ω with the right hierarchical structure, the detailed values are always unreasonably set to zero or a random value due to the poor

ALGORITHM 1: Segment-Generate-Optimize Workflow

Input: Natural Language Input $\mathcal{I}$.
Output: Optimized DT file $\hat{\Omega}$.
New($\mathcal{T}$, V, E)
Segment
$V \leftarrow \mathfrak{L}(P_v, \mathcal{I})$
for each $v_i \in V$ **do**
  $v_p \leftarrow$ parent(v_i)
  append(E, (v_p, v_i))
end for
for each $e_j \in E$ **do**
  $v_{j,p}, v_{j,c} \leftarrow e_j$
  $t_{g,i} \leftarrow \mathfrak{L}\left(P_{s,g}(v_{j,c}, v_{j,p}, \mathcal{I})\right)$
  $t_{o,i} \leftarrow \mathfrak{L}\left(P_{s,o}(R_{j,c}, \mathcal{I})\right)$
  $\mathcal{T}_g[v_{jc}] \leftarrow t_{g,i}$; $\mathcal{T}_o[v_{jc}] \leftarrow t_{o,i}$
end for
Generate
$\bar{\mathcal{T}_g} \leftarrow$ DFSOrder($G, \mathcal{T}_g$)
for each $\bar{t}_{g,i} \in \bar{\mathcal{T}_g}$ **do**
  $j_i = \phi\left(\mathfrak{L}(P_{\phi,g}(t_{g,i}, L_{g,i}))\right)$
  append($\mathcal{J}$, j_i)
end for
Optimize
$\bar{\mathcal{T}_o} \leftarrow$ PostDFSOrder($G, \mathcal{T}_o$)
$\Omega \leftarrow$ Concatenate($\mathcal{J}$)
for each $\bar{t}_{o,i} \in \bar{\mathcal{T}_o}$ **do**
  $\Omega_{i+1} = \phi\left(\mathfrak{L}(P_{\phi,o}(t_{o,i}, L_{o,i})), \Omega_i\right)$
end for
$\hat{\Omega} \leftarrow \Omega_{n+1}$
RETURN $\hat{\Omega}$

mathematical reasoning ability of the language model [12]. Since LLMs can not handle domain-related design tasks, we introduce a set of DCBuild optimized modification methods denoted as $\mathcal{L}_o = \{L_{o,1}, L_{o,2}, L_{o,3}, \ldots, L_{o,i}, \ldots, L_{o,n}\}$ for the facility set V. Based on the optimization subtasks $\mathcal{T}_o$ and the DT file Ω, the optimization process generates a desired optimized DT file $\hat{\Omega}$. We start with the deepest facility as v_1 in graph G and follow the post-order DFS to execute the process as

$$\Omega_1 = \phi\left(\mathfrak{L}(P_{\phi,o}(t_{o,1}, L_{o,1})), \Omega\right), \tag{10}$$

where Ω_1 represents the first time modified Ω related to facility v_1. We can derive the optimized DT file with Equation (10) by

$$\hat{\Omega} = \phi\left(\mathfrak{L}(P_{\phi,o}(t_{o,n}, L_{o,n})), \Omega_n\right). \tag{11}$$

Therefore, the optimized DT file $\hat{\Omega}$ is generated by the initial prompt $\mathcal{I}$ based on Equations (7), (8), (9), and (11). An example of the optimized DT $\hat{\Omega}$ is provided in Appendix B. The SGO workflow algorithm, written as Algorithm 1, enables the automated DT generation and optimized design by the semantic segmentation of the "Planner" and two successive PAL-based generation and optimization processes of the "Caller". As shown in the right part of Figure 6, two optimization sub-tasks are planned during the segment process. It first optimizes each rack model with the server placement principle. Following the post-order DFS, it optimizes the geometry layout of the data hall. Eventually, the SGO workflow synthesizes the final geometric plausible DTs.

5 Data Center Use Cases

In this section, we implement several use cases of ChatDC with a 3D engine to visualize the generated data center DTs. ChatDC provides DTs for industrial practitioners to design the data center layouts via prompts.

5.1 Rule-based Optimized Design for Actuation

ChatDC generates optimized data center DT designs according to user-instructive prompts. We consider data centers different in aisle containment type and room scale. Figure 7 shows four selective data centers with prompts. In terms of hall scale, the given prompts vary from requesting

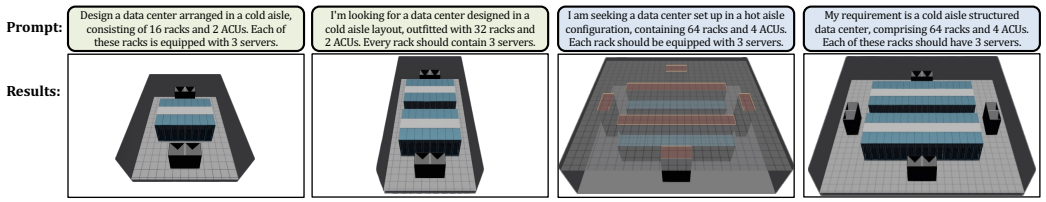

Fig. 7. This paragraph exhibits several use cases enabled by ChatDC. (1) ChatDC can handle prompts for generating data centers at different scales. (2) ChatDC supports mainstream aisle containment types guided by prompts.

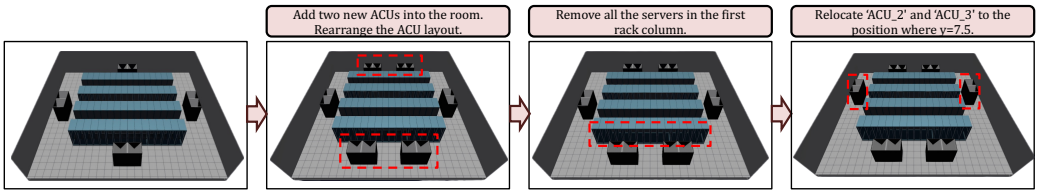

Fig. 8. An example of the state evolution updates of an existing data center within four steps. (1) First, we create two ACUs and rearrange the layout, (2) then we remove the server assets in batch, and (3) finally, we manually change the ACUs' location.

16 racks, and 2 ACUs to comprising 64 racks and 4 ACUs. ChatDC can ensure the geometry correctness and optimum at different scales. There are two mainstream containment types, i.e., cold aisle and hot aisle. When properties are explicitly specified in the prompt, the Planner treats them as fixed and skips issuing any optimization task for that property. With this constraint-aware understanding, ChatDC can recognize the need for different room designs by the Planner and optimize the geometric layout with the respective design.

5.2 Incremental Generation for Digitization

ChatDC allows users to revise an existing data center DT gradually. We show here the changing states of editing a configured DT, including adding new installations, deleting unused assets, and changing facility attributes. Prompts like "*Add two new ACUs into the room.*" can introduce new installations to tackle the potential insufficient cooling capacity. If some unused assets need to be removed, ChatDC supports the deleting operations by prompts like "*Remove all servers in the first rack column*". Operators can also manually modify the attributes of a single facility using prompts like "*Relocate 'ACU_2' and 'ACU_3' to the position where y=7.5*". Figure 8 exhibits several visualized states of a data center DT accommodating 64 racks and 4 ACUs.

5.3 Scalable for High-level Prompts

Some practitioners may want to design a data center by giving some high-level requirements, such as "*I want to build a small data center with limited budgets.*", "*Please build a large data center with a higher cooling efficiency.*". LLMs learned world knowledge before, which also includes data center-related knowledge. Empowered by the erudite LLMs, ChatDC translates the high-level prompts into detailed instructive prompts and generates an optimized DT compliant with the original design intention. It is achieved through extra prompt engineering in $P_{s,g}$. Before the "segment" step of the initial input $\mathcal{J}$, our ChatDC translates the high-level prompts into detailed descriptions, as illustrated in the prompt template in Appendix C. Figure 9 shows two data center DTs generated by ChatDC with high-level and low-level prompts. ChatDC converts the general prompts into instructive ones and generates data center DTs through the SGO.

6 Experimental Evaluation

In this section, we compare the accuracy and efficiency of ChatDC with two baseline methods, our previous framework and ablation settings. We further assess room utilization optimization and the temperature convergence of ChatDC. Lastly, we conduct thermal condition experiments using CFD along with a quantitative analysis.

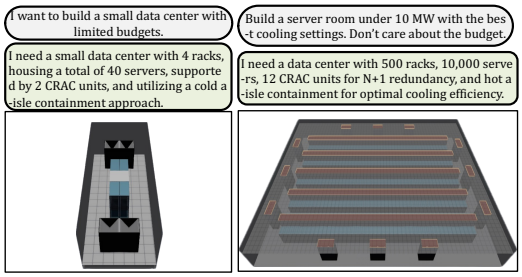

Fig. 9. Cases of using a domain knowledge translator with prompts. The high-level general prompts (upper) are processed by the translator and converted to low-level detailed prompts (lower).

6.1 Baselines

6.1.1 Non-PAL-based Methods. For DT scene description document generation, we compare our solution with two common types of prompting baselines and our previous method [22]:

- **Zero-shot prompting (ZSP)** provides a whole template for generating DT scene description documents without any clues and implements.
- **Few-shot prompting (FSP)** provides several examples for different cases as well as the whole template of the DT scene description document.
- **The Segement-and-Generate (SG)** [21] decomposes the long-text generation into several subtasks and reduces the complexity of the original task.

6.1.2 ChatDC Ablation Settings. We investigate the effectiveness of SGO workflow in handling the generation and optimized design of DTs.

The SGO workflow can be separated into two sub-flows connecting the mentioned three modules, which are denoted as segment-generate and **segment-optimize** (**SO**). Then, we test ChatDC on two settings for ablation study, including (1) ChatDC without using the SGO (w/o SGO), and (2) ChatDC with only the segment-generate and without the SO (w/o SO). The detailed explanations of the two baseline settings are:

- **w/o SGO** directly generates codes from the in-context DCBuild library descriptions;
- **w/o SO** only segments the prompt and generates codes from the in-context DCBuild library descriptions.

Additionally, we maintain an option for ChatDC to utilize in-context examples, which we refer to as binary choices, named **zero-shot** (**ZS**) and **few-shot** (**FS**).

6.1.3 Layout Optimization Policies. We evaluate the room utilization optimization approach in the ChatDC by introducing two baselines, GPT-4 and random.

- The **GPT-4** policy leverages the common-sense reasoning ability of the LLM to select a reasonable layout that meets the constraints in Section 3.2.
- The **random** policy refers to the random selection among all the possible layouts of a data center.

6.2 Evaluation of From-Scratch Generation

6.2.1 Metrics. We use Success Rate to evaluate how well ChatDC generates complete and valid DT files from NL prompts. The generated DT files can be validated in three steps, (1) no syntax error, (2) correct facility number, and (3) no facility collisions.

6.2.2 Experiment Setup. We start by investigating the ChatDC's ability to generate a new data center. First, we compare our PAL-based solution with three common types of prompting baselines in [22] using the previous dataset. Descriptions in the dataset only focus on three levels, the rack, ACU, servers, and room. An example description is *"I want a data hall room with four ACUs and ten racks. Each rack contains two servers"*. We define the prompt complexity C as the number of facilities mentioned in the prompt. Thus, we denote the previous dataset as the "simple" dataset, where each prompt's complexity is $C = 4$. To tackle more comprehensive and complex data center scenarios, we create a new "complex" dataset ($C = 6$) with 200 prompts for testing the generation ability of the SGO workflow. The prompt mentions other facilities, including containment and sensors. Appendix D illustrates one example item in the dataset. The prompt mentions room, rack, ACU, server, containment, opening, and sensor. Regarding the success rate, we compare the ChatDC with the baseline settings in Section 6.1.2. LLMs have varying abilities, fundamentally affecting ChatDC's performance in segmenting prompts and generating codes. We test the SGO workflow with some commercial LLM APIs, including GPT-3.5-Turbo, and GPT-4.

6.2.3 Result and Discussion. ChatDC's SGO workflow enables robust and complete generation, with high success rates across prompt complexities. We start with the "simple" dataset on DT description file generation. The previous SG method achieved a higher success rate of 87% compared with ZSP and FSP, which only have 32% and 64%, respectively. It can also generate more complete files than the baselines, which outperforms 124% and 70% tokens on average. Our latest PAL-based SGO workflow outperforms other baselines with a 98% success rate and 5,231.77 average L_{token} on the "simple" dataset, which indicates ChatDC can precisely generate complete DTs.

Methods	Success Rate	Avg. L_{token}
ZSP	32%	1,656.80
FSP	64%	2,180.15
SG	87%	3,712.02
SGO	**98%**	**5,231.77**

Fig. 10. The success rates and the average token lengths of generated files of ZSP, FSP, SG, and SGO on the original "simple" dataset.

For the "complex" prompts, ChatDC outperforms the baselines under both ZS and FS conditions as it achieves a nearly perfect result of 98% accuracy. Without a complete SGO workflow, ChatDC can only achieve a 65% success rate with FS prompts and an 18% success rate with ZS prompts using GPT-3. When incorporating the PAL-based SG in advance, "w/o SO" greatly raises its ZS success rate to 52% and 43% via GPT-4 and GPT-3. The results in Figure 11 imply that the SGO workflow can significantly enhance the generation success rate compared to the baselines under ZS prompts. Besides, the in-context examples also greatly impact the success rate for optimized DT scene generation. With FS prompts, the success rates can be promoted to 90% for w/o SGO, and 88% for w/o SO using GPT-4 outperforming ZS ChatDC. FS prompts can bolster the performance of w/o SGO and w/o SO with a success rate of 65% and 70%, respectively, under GPT-3. Conversely, their ZS counterparts demonstrate superb performance, even when utilizing GPT-4. The FS ChatDC demonstrates comparable results under GPT-3 and GPT-4. This observation also indicates that the SGO framework can function effectively even with relatively low-performance LLMs. In conclusion, ChatDC significantly improves the success rate on both simple and complex instructive prompts for the DT generation task.

6.3 Evaluation of Incremental Editing

6.3.1 Metrics. We use Fulfilled Rate and Completion Degree to assess ChatDC's ability to handle sequential, multi-step updates to an existing DT. First, the success rate on all steps of an individual test data item is counted as the fulfilled rate of sequential prompts. We test the

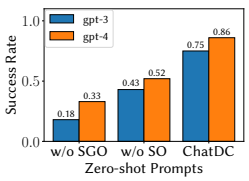

Fig. 11. The success rate of w/o SGO, w/o SO, and ChatDC under FS.

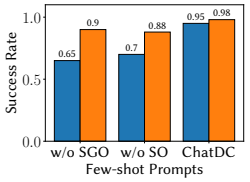

Fig. 12. The success rate of w/o SGO, w/o SO, and ChatDC under ZS.

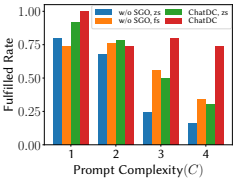

Fig. 13. The fulfilled rate of w/o SGO and ChatDC under ZS and FS.

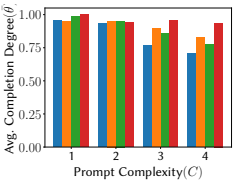

Fig. 14. The average completion degree of w/o SGO and ChatDC in ZS and FS.

incremental generation capacity of ChatDC through different lengths of sequences. Besides, we calculate the average completion degree $\bar{\theta}$ for each improper sequential prompt. If N is the total number of prompts in a test sequence and k is the step at which the model fails (where $1 \leq k \leq N$), the completion degree θ is given by

$$\theta = \begin{cases} \frac{k-1}{N} & \text{if failure occurs at step } k, \\ 1 & \text{if all steps are completed successfully.} \end{cases}$$

For example, if ChatDC is tested with $N = 5$ prompts and fails at step $k = 3$, the completion degree is $\theta = \frac{3-1}{5} = 0.4$. As the sequential prompts possess varying lengths, assessing $\bar{\theta}$ inhibits long sequences from skewing the success rate. We consider the incremental generation based on an optimized mature data center in this experiment setting. The methods include w/o SGO and ChatDC employing both ZS and FS prompts for the ablation experiment. In addition, to better compare with the baselines and exhibit the impact of each module in the SGO workflow, we here employ GPT-3 to conduct the ablation experiments.

6.3.2 Experiment Setup. In practice, creating a structure-complex industrial virtual scene needs multiple steps. Assessing ChatDC in such incremental scenes is vital, where requests are sequentially made and fulfilled to incrementally construct and modify a DT. As the array of 100×5 prompts encompasses a diverse range of facilities, we consider the prompts from single type facility ($C = 1$) to four types ($C = 4$). Therefore, we conduct an ablation study on the ChatDC with these four complexity-varied datasets. An instance of the evaluation datasets with $C = 2$ is in Appendix D.

6.3.3 Result and Discussion. ChatDC demonstrates superior robustness in incremental generation, especially as prompt complexity increases. Figure 13 shows the fulfilled rates of baseline methods and ChatDC on modification tasks for incremental generation. It suggests that the prompt complexity negatively affects the fulfilled rate. Despite this, ChatDC has a better performance compared to the baseline settings. The FS ChatDC still achieves a 74% fulfilled rate when $C = 4$ outperforms the w/o SGO's 34% on FS and 16% on ZS. The in-context examples help w/o SGO to obtain 43% more, compared to the 30% success rate when ChatDC has ZS prompts. To more comprehensively investigate the performance of ChatDC on the DT modification task, we further break down the fulfilled rate into the average completion degree for each prompt sequence. Figure 14 shows that the average completion degree $\bar{\theta}$ degrades with the prompt complexity C. ChatDC has the highest average completion degree up to 100% for the dataset with $C = 1$, which slowly decreases to 93.7% when $C = 4$. In comparison, the $\bar{\theta}$ of w/o SGO decreases rapidly from 95.6% to 70.4% for ZS and from 95.2% to 82.4% for FS.

In conclusion, ChatDC demonstrates its superior performance in data center DT incremental generation with the SGO workflow. The SGO workflow can tackle prompts with higher complexities.

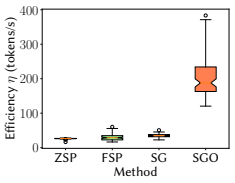

Fig. 15. The efficiency of DT generation by ZSP, FSP, SG, and SGO.

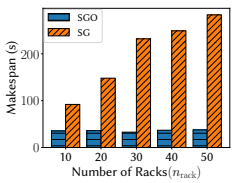

Fig. 16. The average makespan for various numbers of racks by SG and SGO.

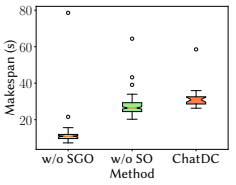

Fig. 17. The average makespan by w/o SGO, w/o SO, and ChatDC.

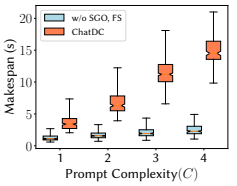

Fig. 18. The average makespan of w/o SGO and ChatDC in different prompt complexities.

This analysis also highlights the relationship between prompt complexity, in-scene facility, and DT context.

6.4 Generation Efficiency

6.4.1 Experiment Setup. To eliminate the inefficacy of LLMs with fewer parameters, we choose GPT-4 to compare the generation efficiency of the baselines and the prior SG framework with our ChatDC. We examine the SGO workflow with the prior non-PAL-based methods using the same test "simple" dataset in [22] that contains 100 data hall NL descriptions. In addition, we illustrate and compare the recorded makespans of different settings of ChatDC on the "complex" prompts in Section 6.2 and the sequential incremental generation prompts in Section 6.3.

6.4.2 Efficiency and Makespan. We define the generation efficiency as

$$\eta = \frac{L_{\text{token}}}{S},$$

where S is the makespan and L_{token} is the generated token length measured by TikToken [32]. Thus, this section reports the *makespan* and *efficiency* across all experimental datasets as the metrics to evaluate the generation efficiency of the ChatDC. The results of non-PAL-based methods and SGO on the DT generation efficiency are shown in Figure 15. Due to the usage of PAL, SGO achieves an impressive generation efficiency by $\bar{\eta} = 204.0$, which is much higher than the non-PAL-based methods in our previous work. The SGO also exhibits its stability for DT at different scales as shown in Figure 16. When the DT scene is small with $n_{\text{rack}} = 10$, the makespans for either the SGO and the SG are at near levels. Whereas when generating a huge data center with $n_{\text{rack}} = 50$, the SGO can finish the task within 35 seconds while the SG takes over 200 seconds. This observation indicates the makespans for tasks executed by the SG proportionally increase along with the n_{rack}. The SGO, on the other hand, generates DT files by codes, which is agnostic to the scale of different data centers.

Then, we compare different settings of ChatDC to investigate the tradeoff between the makespan and the accuracy. Figure 17 shows the makespans of w/o SGO, w/o SO, and ChatDC on executing the prompts in the "complex" dataset. With the SGO workflow, ChatDC can achieve a 48% higher accuracy while only costing 10 more seconds on average. It also indicates the "segment" and "generate" processes cost 50% percent of the time while the "optimize" costs the other half. Figure 18 includes the task makespans of w/o SGO and ChatDC on the incremental generation. Although we reveal the scale-agnosticism of the SGO workflow in Figure 16, it is sensitive to the prompt complexity C. With more facility types mentioned in the given prompts, the makespan for the "segment" process will be proportionally extended. When the prompt complexity $C = 4$, using ChatDC with the SGO workflow can achieve a nearly twice higher fulfilled rate while taking only 10 more seconds to complete the task.

This analysis reveals the superiority of the PAL-based SGO workflow compared to prior non-PAL-based methods. From the above results, the extra makespan of the SGO benefits more in the optimized design and incremental generation.

6.5 Room Layout Optimization

6.5.1 Experiment Setup. According to the data center design considerations in Section 3.2, our ultimate goal is to maximize the room utilization U. This section evaluates the effectiveness of ChatDC in domain-specific versus domain-agnostic generation. We compare ChatDC against the generic GPT-4 reasoning baseline, where the DCBuild optimization step is omitted to provide an ablation study. We examine the outcome based on a two-part evaluation. The first part involves the assessment of the utilization rate and room size of the data center DTs in the "complex" dataset. The second part involves the assessment of temperature convergence of a "small hall" and "large hall" that differ in pre-assigned racks and ACU, respectively. To illustrate, the "small hall" is designed to host 32 racks and 2 ACUs, while the "large hall" is equipped with 128 racks and 10 ACUs. Each rack in both data halls accommodates three servers. We apply the boundary conditions as shown in Figure 19 to evaluate the temperature convergence rate.

Boundary Conditions	Setting
ACU Supply Air Temperature	23 °C
Initial Zone Temperature	28 °C
ACU Supply Air Volume Flow Rate	19.10 $m^3\ s^{-1}$
Server Input Power	1660.0 W
Server Volume Flow Rate	0.02 $m^3\ s^{-1}$

Fig. 19. The setting of boundary conditions.

6.5.2 Room Size and Utilization Rate. First, we test the room size of DTs generated by random, GPT-4, and ChatDC, respectively. The results are shown in Figure 20. With the random selection policy, the median room size of the generated DTs is 348.46 m^2. Using GPT-4 makes less unreasonable DTs, but slightly increases the median room size to 362.48 m^2. For the DTs generated by ChatDC, the room size merely ranged from 78.59 m^2 to 315.48 m^2 averaged at 210.68 m^2.

We then compare the corresponding utilization rates of those DTs, the results are shown in Figure 21. The random policy results in a median room utilization of only 0.14, with a fairly broad range spanning from 0.03 to 0.27. The GPT-4 layout optimization generates a better performance with a median utilization of 0.15 and a dense range compared to the random policy. ChatDC generates DTs with a median utilization of 0.25 and a maximum utilization of 0.30, maintaining steady performance with a minimum utilization of 0.15.

Thus, ChatDC can generate DT halls with the smallest sizes and the highest utilization rates, which aligns with our design principles for room optimization.

6.5.3 Temperature Convergence. We denote $T_z(t)$ as the zone air temperature at time t, m_s as the setpoint of supply air mass flow rate, ρ_{air} as the air density, and P_{IT} as the IT power usage. We measure the temperature by time with a simplified model in EnergyPlus [9], which adopts the nodal model by considering that all CRACs take the same supply settings and the data hall has a uniform spatial temperature distribution. The uniform spatial temperature distribution can be achieved with air containment and thermal-aware load balancing by the **ordinary differential equation** (**ODE**) as

$$\frac{dT_z(t)}{dt} = \frac{m_s(t)}{V_s \rho_{\text{air}}} (T_s(t) - T_z(t)) + \frac{1}{\alpha C_p V_s} P_{\text{IT}}(t), \tag{12}$$

where C_p is the air heat capacity, α is the air recirculation ratio, V_s denotes the data hall volume, m_s and T_s are the supply air mass flow rate and temperature, respectively. We set C_p as 1.006 kJ/(kg · °C), ρ_{air} as 1.16 kg/m^3, and α as 1.3. We assess the temperature convergence of different methods by plotting the temperature change across time. The temperature convergence of the random selection

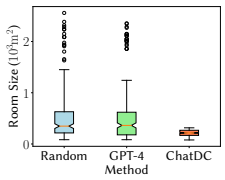

Fig. 20. The room size of the generated DT by random, GPT-4, and ChatDC for room optimization.

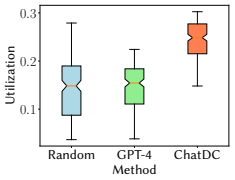

Fig. 21. The utilization of the generated DT by random, GPT-4, and ChatDC for room optimization.

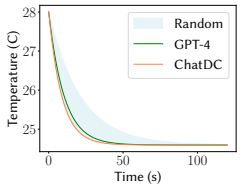

Fig. 22. The temperature convergence of "small hall" in relation to the time with random, GPT-4, and ChatDC.

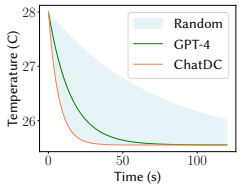

Fig. 23. The temperature convergence of "large hall" in relation to the time with random, GPT-4, and ChatDC.

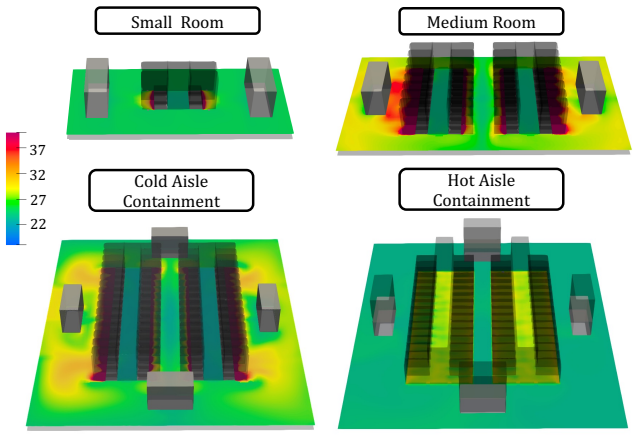

Fig. 24. The CFD simulation results of the generated DTs. (1) The small data center with 16 racks. (2) The medium size data center with cold containment. (3) The large data center has 64 racks designed in the cold aisle containment. (4) The same large data center, but with hot aisle containment.

is depicted as a range using a light blue shaded area, which is presented by the upper bound and the lower bound of all possible layouts.

The results for the "small hall" are shown in Figure 22. ChatDC has the fastest temperature convergence as the room is cooled to the conservative state for around 50 seconds. GPT-4 has a fair performance with a slightly longer time of convergence. The worst case also takes less than 100 seconds to reach the heat conservation. A similar result is due to the small feasible set for the layout optimization. The results for the "large hall" are more comparable with a significantly larger feasible set, as shown in Figure 23. ChatDC again has the fastest temperature convergence as the room is cooled to the conservative state for around 50 seconds. GPT-4 performs much less promising in the "large hall" as it takes nearly 100 seconds to completely cool the room. The worst possible layout takes over 120 seconds to reach the conservative state. To conclude, ChatDC generates the data center DT with minimal room utilization given descriptive prompts. This optimized layout design contributes to the best temperature convergence of the data hall.

6.6 Thermal Performance

6.6.1 CFD Simulation. To confirm the validity of the generated DTs with CFD simulation results, we conduct the experiments based on the use cases guided by low-level prompts and high-level requirements illustrated in Section 5. We set the same setting of boundary conditions for each

data center, listed in Figure 19. The CFD simulation subsequently connects with ChatDC in the pipeline. This generation-simulation pipeline facilitates what-if analysis for data center practitioners. Figure 24 shows the simulation results of four exemplar DTs in Section 5. The optimization design in DCBuild efficiently manages airflow in server rooms of varying sizes, preventing hot spots and enhancing cooling efficiency. Both cold and hot aisle containment designs effectively isolate hot and cold air to ensure efficiency in cooling.

6.6.2 Quantitative Analysis. To condense the wealthy CFD outputs, we employ a popular metric [2], **rack cooling index** (**RCI**) for thermal evaluation. We quantify hot spots by $\mathrm{RCI}_{\mathrm{hi}}$ [16], where $T^{+}_{\mathrm{recommend}}$ and $T^{+}_{\mathrm{allowable}}$ are assigned as the thresholds for the upper limits of the recommended and allowable temperature ranges. It defines the Total Over-Temperature as $\sum_{i=1}^{n} \mathbb{1}(x_i > T^{+}_{\mathrm{recommend}})$ and the Max Allowable Over-Temperature as $\sum_{i=1}^{n} \mathbb{1}(x_i \leq T^{+}_{\mathrm{allowable}})$. Then, the RCI reflects the conditions of rack intake temperatures by

$$\mathrm{RCI}_{\mathrm{hi}} = \left[1 - \frac{\text{Total Over-Temp}}{\text{Max Allowable Over-Temp}}\right] \times 100\%. \tag{13}$$

We calculate $\mathrm{RCI}_{\mathrm{hi}}$ for several DTs in Section 5 based on the CFD simulation results. We set the range of max recommended temperature T^{+}_{r} starting from 23.9 °C to 28.1 °C and fix the max allowable temperature range from 18 °C to 32 °C. Figure 25 shows that all cases reach the acceptable level ($\geq$ 91%) when $T^{+}_{\mathrm{r}} \geq 23.5$. Increasing the standard to 25.2 °C renders all cases at the good level ($\geq$ 96%). When we raise the max recommended temperature to 26.5 °C, all cases fulfill the ideal level of data center cooling demands. ChatDC is scalable for the external translator as indicated in Figure 26. If we set a strict cooling standard by $T^{+}_{\mathrm{recommend}} = 25$, D1 and D2 can still maintain good cooling performances. All four generated DTs are rated as acceptable for normal air-cooled data centers with $T^{+}_{\mathrm{recommend}} = 26$ [34].

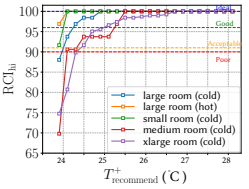

Fig. 25. The relationship between $\mathrm{RCI}_{\mathrm{hi}}$ and T^{+}_{r} on the cases with instructive prompts in Section 5.1.

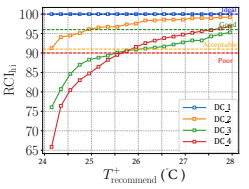

Fig. 26. The relationship between $\mathrm{RCI}_{\mathrm{hi}}$ and T^{+}_{r} on the cases with high-level prompts in Section 5.3.

In conclusion, ChatDC provides the optimization process in the SGO, which regulates the geometry of data center DTs. This analysis reveals that ChatDC's generated DTs can meet the basic data center cooling requirements.

7 Discussion and Future Directions

7.1 Multimodal Data Input

7.1.1 Limitations in NL Prompts. Although the SGO enhanced its scalability for various prompts by converting high-level prompts into instructive prompts, limitations in NL understanding still pose challenges. For instance, the inevitable *hallucination* of LLMs may fail the initial objective and hinder the reliability of generated DT descriptions [48]. The quality of generated DTs can be affected due to improper facility settings and quantities. A purely conversational interactive way may not satisfy the usability requirements of DT systems [20].

7.1.2 Incorporating Multimodal Information. To overcome the inherent ambiguity of language and enhance the fidelity of DT systems, incorporating synergistic data input beyond NL is crucial. Real-time monitoring data offers precise operational and environmental conditions for DT models, reducing reliance on vague language. In a data center, tabular data might include server performance

logs, maintenance records, and energy consumption history. Image data, like visual inspections and thermal imagery, provide additional context about the data center's specifications.

7.2 Generative Design with Physics Constraints

7.2.1 Limitations in Generalization. Nowadays, data centers are not only cooled by air but other advanced cooling techniques. Liquid cooling employs a specialized coolant in a closed loop to dissipate heat from components [17]. Meanwhile, immersive cooling involves submerging electronic components in a non-conductive liquid for efficient cooling [15]. Our approach, using sim-ready assets in DCBuild, cannot generate irregular structures and undefined facilities. Furthermore, generating heterogeneous data centers involves complex physical constraints that cannot be solved by the geometric-aware ChatDC.

*7.2.2 Incorporating **Physics-informed Machine Learning (PIML)**.* To address these limitations, incorporating PIML offers a promising pathway for integrating fundamental physical laws into generative design workflows. By embedding governing equations, PIML ensures the generated designs inherently satisfy thermodynamic constraints. LLMs have recently demonstrated the ability to generate physics-informed code, supporting scientific discovery in complex domains [27]. This synergistic approach has the potential to enhance the ability of LLMs to perform physics-aware generation using diverse sets of generated sim-ready models.

8 Conclusion

In this article, we presented a novel framework that addresses the difficulties in applying LLM to generate data center DTs. This framework integrates domain knowledge to orchestrate LLMs for in-context learning and reasoning abilities on both coding and reasoning tasks, offering a promising approach to automating DT generation for IoA systems.

Our research has demonstrated the potential of LLMs in facilitating the construction of DTs and what-if analysis for industrial environments such as data centers. This will automate the design and optimization processes via prompts, leading to improved operational efficiency and cost savings. By orchestrating the capabilities of LLMs, we have addressed key limitations in prompt comprehension and scene description optimization, paving the way for improved accuracy and reliability in DT generation. Our work represents a significant step towards harnessing the power of LLMs to advance DT generation for the data center industry.

References

[1] Artec 3D. 2023. What is a Digital Twin? Retrieved October 2023 from https://www.artec3d.com/learning-center/digital-twins

[2] A. M. Abbas, A. S. Huzayyin, T. A. Mouneer, and S. A. Nada. 2021. Thermal management and performance enhancement of data centers architectures using aligned/staggered in-row cooling arrangements. *Case Studies in Thermal Engineering* 24 (2021), 100884.

[3] Md Ferdous Alam, Austin Lentsch, Nomi Yu, Sylvia Barmack, Suhin Kim, Daron Acemoglu, John Hart, Simon Johnson, and Faez Ahmed. 2024. From automation to augmentation: Redefining engineering design and manufacturing in the age of NextGen-AI. *An MIT Exploration of Generative AI 1* (2024).

[4] Anthropic. 2025. Model Context Protocol. Retrieved January 2025 from https://modelcontextprotocol.io/

[5] Baanders. 2024. DTDL models - Azure Digital Twins. Retrieved August 2024 from https://learn.microsoft.com/en-us/azure/digital-twins/concepts-models

[6] Philipp Brauner, Manuela Dalibor, Matthias Jarke, Ike Kunze, István Koren, Gerhard Lakemeyer, Martin Liebenberg, Judith Michael, Jan Pennekamp, Christoph Quix, et al. 2022. A computer science perspective on digital transformation in production. *ACM Transactions on Internet of Things* 3, 2 (2022), 1–32.

[7] Angel Chang, Manolis Savva, and Christopher D. Manning. 2014. Learning spatial knowledge for text to 3D scene generation. In *Proceedings of the EMNLP*. 2028–2038.

[8] J. Chen, W. Zhu, and M. M. Ali. 2010. A hybrid simulated annealing algorithm for nonslicing VLSI floorplanning. *IEEE Trans. Syst., Man, Cybern.* 41, 4 (2010), 544–553.
[9] Drury B. Crawley, Linda K. Lawrie, Curtis O. Pedersen, and Frederick C. Winkelmann. 2000. Energy plus: Energy simulation program. *ASHRAE Journal* 42, 4 (2000), 49–56.
[10] Fernanda De La Torre, Cathy Mengying Fang, Han Huang, Andrzej Banburski-Fahey, Judith Amores Fernandez, and Jaron Lanier. 2024. Llmr: Real-time prompting of interactive worlds using large language models. In *ACM CHI*. 1–22.
[11] Weixi Feng, Wanrong Zhu, Tsu-jui Fu, Varun Jampani, Arjun Akula, Xuehai He, Sugato Basu, Xin Eric Wang, and William Yang Wang. 2023. LayoutGPT: compositional visual planning and generation with large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems (New Orleans, LA, USA) (NIPS'23)*. Curran Associates Inc., Red Hook, NY, USA, Article 802, 26 pages.
[12] Simon Frieder, Luca Pinchetti, Alexis Chevalier, Ryan-Rhys Griffiths, Tommaso Salvatori, Thomas Lukasiewicz, Philipp Petersen, and Julius Berner. 2023. Mathematical capabilities of ChatGPT. In *Proceedings of the 37th International Conference on Neural Information Processing Systems (New Orleans, LA, USA) (NIPS'23)*. Curran Associates Inc., Red Hook, NY, USA, Article 1205, 46 pages.
[13] G. Gao, C. Song, T. G. T. A. Bandara, M. Shen, F. Yang, W. Posdorfer, D. Tao, and Y. Wen. 2021. FogChain: A blockchain-based peer-to-peer solar power trading system powered by fog AI. *IEEE Internet Things J.* 9, 7 (2021), 5200–5215.
[14] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *Proceedings of the International Conference on Machine Learning*. PMLR, 10764–10799.
[15] Kawsar Haghshenas, Brian Setz, Yannis Blosch, and Marco Aiello. 2023. Enough hot air: The role of immersion cooling. *Energy Informatics* 6, 1 (2023), 14.
[16] Magnus K. Herrlin et al. 2005. Rack cooling effectiveness in data centers and telecom central offices: The rack cooling index (RCI). *ASHRAE* 111, 2 (2005), 725.
[17] Madhusudan Iyengar, Milnes David, Pritish Parida, Vinod Kamath, Bejoy Kochuparambil, David Graybill, Mark Schultz, Michael Gaynes, Robert Simons, Roger Schmidt, et al. 2012. Server liquid cooling with chiller-less data center design to enable significant energy savings. In *Proceedings of the 2012 28th Annual IEEE SEMI-THERM*. IEEE, 212–223.
[18] Ruoxi Jia, Baihong Jin, Ming Jin, Yuxun Zhou, Ioannis C. Konstantakopoulos, Han Zou, Joyce Kim, Dan Li, Weixi Gu, Reza Arghandeh, et al. 2018. Design automation for smart building systems. *Proc. IEEE* 106, 9 (2018), 1680–1699. DOI: https://doi.org/10.1109/JPROC.2018.2856932
[19] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn. 2018. Digital twin in manufacturing: A categorical literature review and classification. *Ifac-PapersOnline* 51, 11 (2018), 1016–1022.
[20] Emily Kuang, Minghao Li, Mingming Fan, and Kristen Shinohara. 2024. Enhancing UX evaluation through collaboration with conversational AI assistants: Effects of proactive dialogue and timing. In *Proceedings of the ACM CHI*. 1–16.
[21] Jie Li, Yuhui Deng, Rui Wang, Yi Zhou, Hao Feng, Geyong Min, and Xiao Qin. 2024. BTVMP: A burst-aware and thermal-efficient virtual machine placement approach for cloud data centers. *IEEE Transactions on Services Computing* 17, 5 (2024), 2080–2094.
[22] Minghao Li, Ruihang Wang, Xin Zhou, Zhaomeng Zhu, Yonggang Wen, and Rui Tan. 2023. ChatTwin: Toward automated digital twin generation for data center via large language models. In *Proceedings of the ACM BuildSys*. 208–211.
[23] Yuanlong Li, Yonggang Wen, Dacheng Tao, and Kyle Guan. 2019. Transforming cooling optimization for green data center via deep reinforcement learning. *IEEE Trans. Cybern.* 50, 5 (2019), 2002–2013.
[24] Chen Ling, Xujiang Zhao, Jiaying Lu, Chengyuan Deng, Can Zheng, Junxiang Wang, Tanmoy Chowdhury, Yun Li, Hejie Cui, Xuchao Zhang, Tianjiao Zhao, Amit Panalkar, Dhagash Mehta, Stefano Pasquali, Wei Cheng, Haoyu Wang, Yanchi Liu, Zhengzhang Chen, Haifeng Chen, Chris White, Quanquan Gu, Jian Pei, Carl Yang, and Liang Zhao. 2025. Domain specialization as the key to make large language models Disruptive: A comprehensive survey. *ACM Comput. Surv.* 58, 3, Article 79 (Oct. 2025), 39.
[25] Bang Liu, Xinfeng Li, Jiayi Zhang, Jinlin Wang, Tanjin He, Sirui Hong, Hongzhang Liu, Shaokun Zhang, Kaitao Song, Kunlun Zhu, et al. 2025. Advances and challenges in foundation agents: From brain-inspired intelligence to evolutionary, collaborative, and safe systems. arXiv:2504.01990. Retrieved from https://arxiv.org/abs/2504.01990
[26] Hexu Liu, Gurjeet Singh, Ming Lu, Ahmed Bouferguene, and Mohamed Al-Hussein. 2018. BIM-based automated design and planning for boarding of light-frame residential buildings. *Automation in Construction* 89 (2018), 235–249. https://www.sciencedirect.com/science/article/pii/S0926580518301018
[27] Pingchuan Ma, Tsun-Hsuan Wang, Minghao Guo, Zhiqing Sun, Joshua B Tenenbaum, Daniela Rus, Chuang Gan, and Wojciech Matusik. 2024. LLM and simulation as bilevel optimizers: A new paradigm to advance physical scientific discovery. In *Proceedings of the 41st International Conference on Machine Learning*. 33940–33962.
[28] Rui Ma, Akshay Gadi Patil, Matthew Fisher, Manyi Li, Sören Pirk, Binh-Son Hua, Sai-Kit Yeung, Xin Tong, Leonidas Guibas, and Hao Zhang. 2018. Language-driven synthesis of 3D scenes from scene databases. *ACM TOG* 37, 6 (2018), 1–16.

[29] NVIDIA. 2023. NVIDIA Omniverse. Retrieved October 2023 from https://www.nvidia.com/en-sg/omniverse/
[30] NVIDIA Corporation. 2023. SimReady Assets. NVIDIA Omniverse Developer Portal. Retrieved October 2023 from https://developer.nvidia.com/omniverse/simready-assets
[31] OpenAI. 2023. Function calling. Retrieved October 2023 from https://platform.openai.com/docs/guides/function-calling
[32] OpenAI. 2023. Tiktoken, a fast BPE tokeniser for use with OpenAI's models. Retrieved October 2023 from https://github.com/openai/tiktoken
[33] Konstantinos Papamichael, John LaPorta, and Hannah Chauvet. 1997. Building design advisor: Automated integration of multiple simulation tools. *Automation in construction* 6, 4 (1997), 341–352.
[34] Suhas V. Patankar. 2010. Airflow and cooling in a data center. *Journal of Heat Transfer* 132, 7 (04 2010), 073001.
[35] Yongyi Ran, Han Hu, Yonggang Wen, and Xin Zhou. 2022. Optimizing energy efficiency for data center via parameterized deep reinforcement learning. *IEEE Trans. Serv. Comput.* 16, 2 (2022), 1310–1323.
[36] Yongyi Ran, Xin Zhou, Han Hu, and Yonggang Wen. 2022. Optimizing data center energy efficiency via event-driven deep reinforcement learning. *IEEE Trans. Serv. Comput.* 16, 2 (2022), 1296–1309.
[37] Timo Schick, Jane Dwivedi-Yu, Roberto Dessí, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Proceedings of the 37th International Conference on Neural Information Processing Systems (New Orleans, LA, USA) (NIPS'23)*. Curran Associates Inc., Red Hook, NY, USA, Article 2997, 13 pages.
[38] Roger Schmidt and Madhusudan Iyengar. 2005. Effect of data center layout on rack inlet air temperatures. In *Proceedings of the InterPACK*. 517–525.
[39] Roger R. Schmidt, Ethan E. Cruz, and M. Iyengar. 2005. Challenges of data center thermal management. *IBM Journal of Research and Development* 49, 4.5 (2005), 709–723.
[40] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y. Wu, et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. arXiv:2402.03300. Retrieved from https://arxiv.org/abs/2402.03300
[41] Significant Gravitas. [n. d.]. *AutoGPT*. Retrieved August 2024 from https://github.com/Significant-Gravitas/AutoGPT
[42] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. 2023. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2998–3009.
[43] Chunyi Sun, Junlin Han, Weijian Deng, Xinlong Wang, Zishan Qin, and Stephen Gould. 2025. 3D-GPT: Procedural 3D modeling with large language models. In *2025 International Conference on 3D Vision (3DV)*. 1253–1263.
[44] Fan-Yun Sun, Weiyu Liu, Siyi Gu, Dylan Lim, Goutam Bhat, Federico Tombari, Manling Li, Nick Haber, and Jiajun Wu. 2025. LayoutVLM: Differentiable optimization of 3D layout via vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 29469–29478.
[45] Fei Tao, He Zhang, Ang Liu, and Andrew YC Nee. 2018. Digital twin in industry: State-of-the-art. *IEEE Transactions on Industrial Informatics* 15, 4 (2018), 2405–2415.
[46] Wenfeng Xia, Peng Zhao, Yonggang Wen, and Haiyong Xie. 2016. A survey on data center networking (DCN): Infrastructure and operations. *IEEE Commun. Surv. Tutorials* 19, 1 (2016), 640–656.
[47] Fengli Xu, Qianyue Hao, Zefang Zong, Jingwei Wang, Yunke Zhang, Jingyi Wang, Xiaochong Lan, Jiahui Gong, Tianjian Ouyang, Fanjin Meng, et al. 2025. Towards large reasoning models: A survey of reinforced reasoning with large language models. arXiv:2501.09686. Retrieved from https://arxiv.org/abs/2501.09686
[48] Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. 2024. Hallucination is inevitable: An innate limitation of large language models. arXiv:2401.11817. Retrieved from https://arxiv.org/abs/2401.11817
[49] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
[50] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large language models are human-level prompt engineers. In *International Conference on Learning Representations (ICLR)*.

Appendices

A Example Description File Fragments for Individual Assets

Listing 1. Example DT fragment representing the location and size of an ACU asset.

```
"ACU_1": {
    "location": {
        "x": 0,
        "y": 0,
        "z": 0
    },
    "size": {
        "x": 0,
        "y": 0,
        "z": 0
    }
}
```

Listing 2. Example DT fragment representing a rack and its internal server configuration.

```
"Rack_0": {
    "geometry": {
        "model": "RackType_A",
        "location": {"x": 0, "y": 0, "z": 0 },
        "orientation": 0,
        "hasBlankingPanel": true
    },
    "constructions": {
        "servers": {
            "Rack_0_Server_0": {
                "geometry": {
                    "model": "1U_Server",
                    "slotPosition": 0
                },
                "cooling": {
                    "model": "CPU_Server"
                },
                "power": {
                    "model": "PowerModel_A"
                }
            },
            "Rack_0_Server_1": {
                "geometry": {
                    "model": "1U_Server",
                    "slotPosition": 0
                },
                "cooling": {
                    "model": "CPU_Server"
                },
                "power": {
                    "model": "PowerModel_A"
                }
            },
        }
    }
}
```

B Optimized DT File for a Virtual Data Hall

Below we show an excerpt of the generated DT file, with some repetitive details omitted for brevity.

Listing 3. An optimized DT file describing a virtual data hall with 4 ACUs, 64 racks, and server placements.

```
{
  "meta": {
    "description": "A virtual data hall that is built with 4 acus, 64 racks.
         Each rack has 3 servers. The room layout is in the cold aisle."
  },
  "geometry": {
    "height": 3.5,
    "plane": [ { "x": 0, "y": 0, "z": 0 }, ... ]
  },
  "constructions": {
    "acus": {
      "CRAC_0": {
        "geometry": {
          "model": "ACU_COLD_AISLE",
          "orientation": -90,
          "location": { "x": 6.68, "y": 1.995, "z": 0.45 }
        },
        "cooling": { "model": "ACCPU_1" }
      },
      ...
    },
    "racks": {
      "Rack_0": {
        "geometry": {
          "model": "RackType_A",
          "location": { "x": 3.595, "y": 4.195, "z": 0.45 },
          "orientation": 180
        },
        "constructions": {
          "servers": {
            "Rack_0_Server_0": { "geometry": { "model": "1U_Server", "
                slotPosition": 1 } },
            ...
          }
        }
      },
      ...
    },
    "raisedFloor": {
      "geometry": {
        "height": 0.45,
        "openings": {
          "opening_CRAC_0": { "location": { ... }, "size": { ... } },
          ...
        }
      }
    }
  },
  ...
}
```

C Template for High-level Prompts

Listing 4. The prompt template for translating the high-level prompts into detailed descriptions.

```
You are a data center configuration assistant.
User provides high-level goals of a data center, and you generate detailed
    instructive prompts to describe its configuration.

# KNOWN PARAMETERS:
- Total racks: [R]
- Total servers: [S]
- Total CRACs: [C]
- Sensor count: [N]
- Containment type: [T]  # e.g., hot aisle, cold aisle

# USER QUERY:
{input}

# TASK:
1. Analyze the query's technical requirements
2. Identify which parameters need derivation
3. Generate specific values through step-by-step reasoning
4. Summarize in one sentence within ``` description {} ```.

Example:
``` description
I need a data center room with 25 racks, 3 cracs, 6 servers distributed in
    each rack. The server room is in cold aisle containment.
```

REASONING:
```

D Evaluation Prompt Datasets

Listing 1: An example from the evaluation dataset for from-scratch generation.

```
{
    "rack": 12,
    "crac": 2,
    "server": 4,
    "sensor": 9,
    "containment": "hot",
     "input": "The data center room contains 12 racks, 2 CRAC units, and 9 sensors. Each rack contains 4
        servers. The room is designed in hot aisle containment."
}
```

Listing 2: An example from the evaluation dataset for incremental editing (complexity = 2).

```
{
    "operations": [{
            "Operation": "delete_by_name",
            "Facility": "rack",
            "Arguments": {"name": "Rack_0"}
        },{
            "Operation": "create",
            "Facility": "sensor",
            "Arguments": {"quantity": 3}
    }],
    "input": "Delete 'Rack_0' and create 3 sensors in the room."
}
```

Received 28 February 2025; revised 11 August 2025; accepted 4 October 2025