SN Computer Science (2024) 5:589
https://doi.org/10.1007/s42979-024-02907-0

SURVEY ARTICLE

From Algorithms to Architecture: Computational Methods for House Floorplan Generation

Azmeraw Bekele Yenew[1] · **Beakal Gizachew Assefa**[1]

Received: 30 December 2023 / Accepted: 17 April 2024

Abstract

House floorplan generation entails crafting efficient spatial layouts within buildings, harmonizing functionality, aesthetics, and usability. The automation of this process is pivotal, expediting design timelines, reducing errors, conserving resources, and facilitating swift exploration of diverse design alternatives for optimal functionality and aesthetics. Nonetheless, the field grapples with inherent challenges, including the provision of diverse layouts to accommodate varied preferences, striking a balance between visual and functional realism, meeting customization demands, and aligning with architectural constraints. In this article, we delve into the transformative impact of computational methods on house floorplan generation. Our study offers a nuanced review and innovative categorization of computational techniques, distinguishing between procedural and deep generative learning approaches. Additionally, we examine representation methods and their interactive capabilities, providing a comprehensive analysis of the advancements, merits, and limitations of contemporary techniques. Furthermore, we critically assess unresolved challenges and delineate promising avenues for future research in computational-based floorplan generation.

Keywords Generative adversarial networks · Floorplan generation · Deep learning · Interactivity · Procedural methods · Representation

Introduction

In the realm of architectural design, interior planning, and spatial optimization, the creation of efficient floorplans stands as a foundational challenge. The arrangement of rooms, corridors, and open spaces within a building significantly influences its functionality, aesthetics, and overall usability. Over the years, architects and designers have employed manual techniques to create floorplans, relying on their expertise and creativity to meet the specific needs of clients and occupants [1]. However, the digital era has opened up new possibilities with advanced computer-aided design (CAD) (Carpo 2017) and the fusion of computational algorithms. This has sparked a revolution in floorplan generation, with various techniques ranging from Procedural Methods [2–6] to deep generative models [7–9]. Procedural Methods is an approach that relies on algorithms and predefined rules to automatically generate floorplans for buildings or interior spaces [10]. While procedural methods offer a structured approach to floorplan generation, they often struggle to capture the nuanced design elements and contextual relevance present in modern architecture. Another cutting-edge approach that leverages artificial intelligence, particularly deep learning techniques to automatically create floorplans is deep generative models [11]. These models use complex neural network architectures to learn patterns and relationships from existing floorplan data and then generate new, coherent, and contextually relevant floor plans. This technology has gained significant attention in recent years due to its ability to produce highly realistic and diverse designs. Generative models in floorplan generation encompass Generative Adversarial Networks (GANs) [12–18] by a generator-discriminator interplay, Diffusion generative model [19] by iterative diffusions, Autoregressive Models [20] for sequential generation, and other deep learning techniques [21–23].

✉ Azmeraw Bekele Yenew
azmeraw.bekele@aait.edu.et

✉ Beakal Gizachew Assefa
beakal.gizachew@aait.edu.et

[1] School of Information Technology and Engineering, Addis Ababa University, Addis Ababa, Ethiopia

Published online: 27 May 2024

The emerging landscape of computational-based architectural software presents a diverse array of tools tailored to streamline and enhance various stages of the design process. Qbiq stands out as an efficient planning solution for office spaces, providing architects with three 'Test fit' alternatives backed by analytical furniture plans and immersive 3D virtual tours. Its performance analysis reports offer insights into crucial factors like privacy, daylight access, and density ratios, facilitating rapid decision-making toward optimal layouts. TestFit, on the other hand, focuses on feasibility studies and urban planning, automating tedious tasks like counting housing units and parking spots to empower architects to delve deeper into creativity. Its real-time rendering capabilities and integration with popular design programs underscore its potential to revolutionize large-scale projects. Meanwhile, Aino transforms site analysis data into actionable maps, Finch facilitates data-driven architectural plans, CONIX.AI offers tailored solutions for residential designs, and Laiout emerges as a promising tool for early-stage design projects, showcasing the diverse applications and advancements in AI-driven architecture.

The significant benefits offered by those computational methods in the floorplan creation process strongly underscore the urgency and relevance of computational methods in contemporary architectural practices. This groundbreaking technology enhances efficiency by automating repetitive tasks such as drafting and 3D modeling [24], freeing architects to focus on higher-level design decisions and optimize project timelines [25]. Moreover, computational methods enable predictive analysis, allowing architects to anticipate building performance, energy consumption, and structural integrity, facilitating informed design choices that prioritize sustainability and efficiency [26]. By integrating computational methods, architects can leverage big data to gain a deeper understanding of urban patterns and trends, ultimately creating structures that are more attuned to their environment and inhabitants [27]. With the rapid pace of urbanization and the growing demand for sustainable, efficient design solutions, architects are increasingly turning to computational tools to meet these complex challenges effectively. It enables architects to expedite design iterations, optimize layouts, and enhance design quality while simultaneously reducing labor costs. This not only streamlines the design process but also empowers architects to respond more efficiently to the evolving needs of clients, communities, and the environment. As such, the adoption of computational methods represents a fundamental shift in architectural practices, emphasizing the critical importance of embracing technological advancements to ensure the continued relevance and success of the profession in an ever-changing world.

This advancement addresses various user needs and industry trends by revolutionizing the architectural and real estate sectors. By harnessing different algorithms, floorplan generation tools have become remarkably efficient, enabling the creation of accurate floorplans in a fraction of the time and effort required by manual drafting [28]. These tools offer a high degree of customization and flexibility, accommodating various design preferences and specific requirements. Moreover, it optimizes space utilization, ensuring that floorplans are practical and well-designed by considering factors such as traffic flow and natural lighting [29]. Additionally, this method provides valuable design assistance and inspiration by analyzing vast amounts of data, offering creative suggestions, and adhering to industry best practices [20]. Collaboration is enhanced through computational methods, facilitating the involvement of multiple stakeholders and enabling iterative design processes [30]. Lastly, the accessibility and democratization of AI-powered tools have revolutionized the field, empowering individuals without formal training to engage in architectural design.

While notable progress has been achieved in the field of floorplan generation, a set of persistent challenges remains to be addressed. These encompass ensuring the diversity of generated floorplans[15, 16], maintaining functional realism [20] and visual realism [19], addressing interactivity concerns for user engagement [22], and effectively tackling customization complexities [20]. Another challenge within floorplan optimization stems from the adaptable nature of walls and rooms, which frequently lack predetermined dimensions. This inherent variability can pose difficulties in achieving convergence for the subsequent optimization model[12]. Overcoming these hurdles is essential to harness the full potential of floorplan generation, enabling it to cater to a wide range of preferences, enhance user experiences, and provide practical yet imaginative solutions within the realm of architectural design.

Figure 1 presents the taxonomy used in this comprehensive review, consisting of three core divisions: Constraint, Representation, and Methods. Input constraint or interactivity refers to how a user or designer can interact with or provide input to the floorplan generation system. This interaction can take various forms such as images [18, 31], text [23], sketches [12], or bubble diagrams [14–16, 19], and it significantly impacts the final output of the floorplan generation process. Another crucial category is representation methods [32], which involves translating input data into a more abstract and meaningful format that facilitates comprehension and seamless interaction for the models. Lastly, the Methods category encompasses the specific techniques, algorithms, and approaches used to create floorplan layouts, such as procedural, and/or deep learning methods. These methods play a critical role in determining the quality, efficiency, and variety of the generated floorplans.

Although there is a shortage of comprehensive and systematic reviews in the realm of floorplan generation through computational techniques, Table 1 provides an overview of

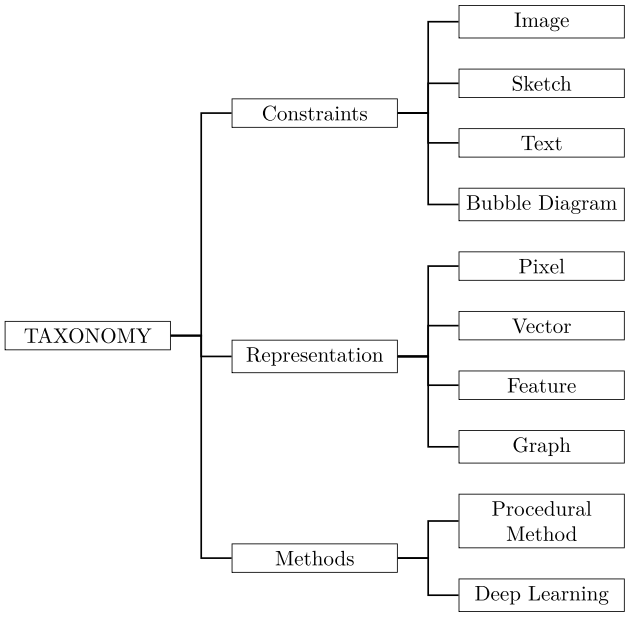

Fig. 1 Taxonomy of floorplan generation techniques

survey studies focused on floorplan generation, along with their corresponding objectives. The columns of the table indicate the theme and scope, and the symbol "✓" tells the topic is covered in the survey. Yong et al. [28] endeavor to provide an encompassing assessment of floorplan generation through generative models, while also exploring their applications. In contrast to their approach, our review seeks to delve into the impact of interactivity on floorplan generation and the role of representation in influencing generative models. Diverging from their focus solely on generative models, our review aims to encompass procedural as well. Within each subcategory, we present a comprehensive analysis of general algorithms and pivotal attributes in cutting-edge solutions, accompanied by their respective advantages and disadvantages. Moreover, we undertake a comparative assessment of these methods alongside models. As the main contributions in this article, we:

- Investigate the role of interactivity in floorplan generation, including how user engagement influences design outcomes and how interactive techniques enhance user-centric solutions.
- Analyze the significance of representation methods in floorplan generation, exploring how different ways of encoding input data affect the quality and diversity of generated floorplans.
- Present a taxonomy of Floorplan Generation methods into subcategories of Procedural, and deep learning methods.
- Conduct a comprehensive evaluation of the discussed methods, comparing their strengths and weaknesses in terms of output quality, diversity, user engagement, and computational efficiency.
- Identifying existing challenges and limitations in floorplan generation such as handling irregular shapes, maintaining user customization, or ensuring compatibility with architectural constraints.

In the upcoming parts of this review, we'll explore interactivity in "User Interaction and Engagement" section and representation techniques in "Representation Methods". In "Computational Methods" section will provide an in-depth look into different approaches used for floorplan generation, covering procedural, and deep learning methods. We'll analyze each category's main contributions, strengths, and limitations. Moving on, in "The Importance of Generating Floorplans Using Computational Method" section discusses the importance, and in "Open Challenges and Possible Solutions" will address challenges and potential future paths. Lastly, in "Future research Directions" section discusses the future directions and "Conclusion" section will offer the conclusion for the entire review.

User Interaction and Engagement

Interactivity has ushered in a new era of creative possibilities and practical applications. This multifaceted concept spans various types of engagement, each tailored to distinct artistic or functional objectives. From direct manipulation to guided customization, interactive techniques allow users to influence the artistic process and contribute their unique vision. For instance, within the realm of image generation, interactivity enables the transformation of various inputs such as images [33, 34], text descriptions [35, 36], and other modalities, thereby bridging the chasm between different modes of expression. Similarly, the converse holds with text generation, where users' inputs in the form of text [37–39] or images [40] can be translated into coherent textual narratives, uniting the realms of language and visual representation. This symphony of interactivity thus amplifies creativity, forging connections

Table 1 Comparison of survey works on floorplan generation

Name	Theme	Interactivity	Representation	PM	DL
Yong et al, 2023 [28]	Deep learning				✓
Our Survey	Computational methods	✓	✓	✓	✓

Note: PM represents procedural methods, while DL signifies Deep learning

between different forms of communication and enabling an enriched exchange of ideas and artistic visions.

Types of Interactivity

The field of floorplan generation has experienced a significant transformation with the advent of interactivity, enabling active user participation in the design process. This innovative approach empowers architects, designers, and homeowners to actively shape and refine their ideal floorplans using user-friendly interfaces and receiving real-time feedback. While floorplans are commonly generated automatically without human intervention [41], interactivity further enhances the process by allowing users to effortlessly incorporate, customize, and visualize rooms and components according to their preferences. In the process of generating floor plans, four types of input constraints play a crucial role: image, text, bubble diagram, and sketch. These input constraints effectively govern the layout and design considerations, allowing for a comprehensive approach to floor plan creation.

Image-to-Floorplan

It is the transformation of an input image, such as a reference image (with normal boundaries [12] or masked boundaries [18, 42]) into a corresponding output floorplan layout. This transformation utilizes computational techniques to automate the conversion process while optimizing spatial arrangement, architectural elements, and aesthetic considerations.

Text-to-Floorplan

It is a transformation of written descriptions of a building's layout, room arrangements, dimensions, and other architectural details into a visual representation of the floorplan [23]. This process involves interpreting the natural language description, understanding spatial relationships and design elements, and then creating a digital floorplan that accurately reflects the described space.

Buble Diagram-to-Floorplan

Generating a floor plan from a bubble diagram involves translating the conceptual layout and relationships depicted in the bubble diagram into a detailed architectural floorplan [14–16, 19, 20]. A bubble diagram is a rough sketch or diagram that uses simple geometric shapes (often circles or bubbles) to represent different spaces or functional areas within a building [43]. These bubbles are interconnected with lines to show the flow and adjacency between spaces. The process of generating a floorplan from a bubble diagram typically entails refining the rough spatial arrangement, determining specific room dimensions, adding walls, doors, windows, and other architectural elements, and ultimately transforming the abstract representation of the diagram into a practical and functional architectural layout. This conversion process helps architects and designers turn initial conceptual ideas into concrete designs that can be further developed and realized.

Sketch-to-Floorplan

Generating a floorplan from a sketch drawing refers to the process of using computational algorithms, to convert a rough sketch of a building's boundary into a more precise and detailed digital floorplan [22]. This involves analyzing the lines, and shapes and translating them into accurately scaled measurements, and architectural elements.

Discussion

Table 2 provides an overview of various tools that effectively control different aspects of control room design. It categorizes the user experience into three levels of intuitiveness: "High" represents a capability that is effective and user-friendly, ensuring a seamless and intuitive experience. The term "Moderate" indicates a medium level of

Table 2 Comparisons based on Interactivity

Constraint	Speed	Customize	Flexibility	Room size	Room shape	Room Type	Room adjacency	Layout Boundary
Text [23]	Moderate	High	High	✓	✗	✓	✓	✗
Sketch [12]	Moderate	Moderate	High	–	–	–	–	✓
Image [13, 17, 18, 31, 42, 44]	High	Low	Low	–	–	–	–	✓
Bubble_Diagram [14–17, 19, 20]	Low	High	Moderate	–	✗	✓	✓	✗

capability, implying that there may be some complexities or considerations to take into account while using the tool. On the other hand, the term "Low" suggests a lack of capability, meaning that the tool is not equipped to address the specific issue or requirement. The table also utilizes symbols to indicate the capability of each input constraint in addressing specific issues: a "✓" signifies the presence of the capability, "✓" denotes a moderate capability with some complexities, and "-" represents the absence of the capability.

In addition, the input modalities for floorplan generation possess several fundamental and desirable properties. Speed is a crucial aspect, determining the efficiency of generating floor plans using a particular method. Customization options play a significant role, allowing users to personalize and tailor the generated floor plans to their specific preferences and needs. This flexibility encourages creativity and exploration, enabling users to experiment with diverse design ideas and approaches.

Geometric properties, such as room size and shape, are important spatial characteristics that define the layout and arrangement of elements within a floorplan. Room size influences functionality, furniture placement, and overall comfort, while room shape affects aesthetics and flow of movement. Semantic properties encompass attributes like the number of rooms and their types, enabling dynamic adjustments and assigning specific functions or purposes to each room. This adaptability allows users to create floor plans that align with their intended uses and requirements.

Topological properties focus on the abstract arrangement and connectivity of spaces, particularly room adjacency. The flow and accessibility between different areas are determined by the relationships and connections between rooms. Lastly, the layout boundary establishes spatial boundaries, serving as a framework for organizing the floorplan and defining its overall shape and size.

Text-based floorplan generation, as shown in Table 2, empowers users to express their floorplan preferences through descriptive textual input, covering room quantities, sizes, types, and spatial relationships. The system interprets this textual data to create corresponding floorplans, showcasing its impressive capabilities. However, this approach faces challenges in accurately representing irregular room shapes and layout boundaries. Moreover, text-based representations have their limitations, including potential ambiguities and misinterpretations in the input [45], reduced contextual understanding for longer inputs [46], and difficulties in handling unique or domain-specific terminologies. Consequently, these factors can impact the overall accuracy of text-based floorplan generation and make the moderate ease of use an additional consideration. Another constraint that poses difficulties in terms of usability is sketching, as discussed in[12]. Sketching, by its nature, requires experience or a natural talent, which contributes to a lower level of ease of use. Additionally, controlling each room's size, shape, type, and number through sketching alters the concept of generation. Therefore, this method is primarily important for controlling the boundary of the generated floorplan, rather than for the overall generation process.

Fortunately, there is an alternative method that overcomes these challenges by directly generating floorplans from input images [13, 17, 18, 31, 42, 44]. This method allows users to insert masked or normal boundaries as controllable constraints. However, it is important to note that, similar to sketching, this method may also encounter difficulties in precisely controlling room size, shape, number, and sizes. Nonetheless, compared to text-based floorplan generation or sketching, the direct generation from input images approach offers a more visually-oriented and accessible means of expressing floorplan preferences. It provides a starting point that can be further refined and adjusted to meet specific requirements.

Another commonly employed constraint in floorplan generation is the bubble diagram, as discussed in [14–16, 19, 20]. The bubble diagram approach enables users to experiment with different room arrangements, room types, and the number of rooms, facilitating a process of creative exploration. It offers a quick and easy way to visualize and iterate on floorplan designs. However, one limitation of the bubble diagram method is its inability to effectively represent irregular room shapes and articulate precise room sizes. The focus of the bubble diagram is primarily on the overall layout and spatial relationships between rooms, rather than capturing intricate details of individual room shapes or specific sizes. Despite this limitation, the bubble diagram remains a valuable tool for generating floorplan concepts and facilitating quick iterations. It allows users to explore various room arrangements and types, providing a foundation for further refinement and customization in terms of room shapes and sizes through subsequent steps or alternative approaches in the floorplan generation process.

In this analysis, we examine the impact of interactivity from the users' perspective, considering its implications on speed, customization, and flexibility as shown in table 2. The speed ratings are based on how quickly users can make floor plans using each method. With image-based generation, which receives a high rating, users simply need to insert existing plans, making the process relatively fast. Sketch-based generation, rated moderate, can be relatively fast for skilled users who can quickly sketch their ideas, which are then interpreted into plans by the model. Text-based generation involves describing plans in writing, which can be done pretty quickly if users are good at explaining their ideas. Bubble diagrams take more time because users have to place bubbles to represent spaces and then refine them iteratively. These speeds can vary depending on factors like

how complex the design is and how skilled users are with each method.

The extent of customization options available varies across the different methods of floor plan generation. Image-based generation, rated as limited, typically offers fewer opportunities for customization due to its reliance on existing plans, which may have limited flexibility for modification. In contrast, text-based generation, rated high, provides extensive customization options as users can specify detailed design parameters and preferences in written form, granting precise control over the generated floor plans. Similarly, bubble diagrams, also rated high, afford ample customization opportunities as users can iteratively refine the layout by adjusting the placement and size of bubbles to accurately represent their design intentions. Sketch-based generation, rated moderate, offers a moderate level of customization, allowing users to sketch out their ideas with some flexibility. However, this may be constrained by the interpretative capabilities of the generative model. These ratings reflect the spectrum of customization options available within each method, with text-based and bubble diagram methods offering the highest degree of flexibility and control over the resulting floor plans.

Text-based generation and sketch-based generation methods are attributed with a high flexibility rating due to their capacity to offer users a broad spectrum of options and adaptability. These methods facilitate creativity and exploration by allowing users to specify detailed design parameters in textual or visual form, thereby enabling diverse design ideas and approaches to be easily explored and experimented with. Conversely, bubble diagrams, rated moderately for flexibility, provide users with some degree of adaptability as they can manipulate the placement and size of bubbles to represent various spatial relationships within the constraints of the diagram format. However, image-based generation is assigned a low flexibility rating primarily because users are constrained to working with existing plans, limiting the scope for innovative experimentation and creative exploration. In summary, while text and sketch-based methods afford significant flexibility, bubble diagrams offer a moderate level of adaptability, and image-based generation presents the least amount of flexibility in accommodating diverse design approaches.

Notably, textual descriptions have emerged as a robust method for determining optimal control room sizes, offering a precise and straightforward approach. However, when considering control number of rooms and types, the combined use of text and bubble diagrams has proven to be particularly advantageous, facilitating a comprehensive understanding of these crucial factors. Additionally, the use of images and sketches has been found to be instrumental in defining the boundaries and aiding in visualizing the extent of influence. Furthermore, the survey underscores that boundary(sketch or image) and graph-based methods emerge as ideal approaches for effectively managing the physical boundaries of floorplans and the spatial relationship between rooms [22]. While dealing with the challenge of room shape regulation, it is acknowledged that managing irregular room shapes can be intricate.

Representation Methods

The representation method refers to how the raw input data is transformed, described, and encoded into a format that can be effectively processed and used by computational models [32]. It's a way of transforming the architectural elements (room type, size, position) and spatial relationships of the house into a format that can be manipulated and processed by algorithms. Understanding representation methods is pivotal in floorplan generation as they dictate how diverse forms of input data are translated into structured formats that computational models can comprehend, ultimately influencing the accuracy, creativity, and efficiency of the generated floorplans.

Types

The choice of representation method has a significant impact on the performance of generative models. A well-chosen representation can make the underlying patterns and relationships in the data more apparent and relevant to the task at hand. There are several types of representation methods used to capture different aspects of the input data and suited for specific types of tasks:

Pixel-Based Representation

Pixel-based representations are the most direct way to represent data [47]. Each pixel is treated as a separate data point, and the color or intensity values of pixels are directly manipulated to generate images. While pixel-based methods are straightforward, they may lack a high-level semantic understanding of the content they generate.

Vector-Based Representation

Vector-based representations use vector space mathematics to represent data [48]. These representations often involve encoding various floorplan attributes, such as color, shape, texture, room type, and more, into vectors that can be manipulated to generate.

Feature-Based Representation

In this approach, images are represented as a set of high-level features or attributes, extracted using techniques like

convolutional neural networks (CNNs) [49] or other feature extraction methods. These features capture meaningful information from the images and can be used to generate new images with specific attributes.

Graph-Based Representation

Graph-based representation is a method used to model and depict complex relationships between entities using a graph structure. A graph is a mathematical structure that consists of nodes (vertices) and edges (connections) between those nodes [50]. Nodes signify distinct entities with associated attributes, while edges denote connections or relationships between nodes. This approach adeptly captures intricate relationships in diverse datasets, enabling comprehensive analysis and visualization across various fields.

Discussion

When it comes to generating intricate floorplans, selecting the right representation method is a pivotal decision. Floorplans are inherently structured and geometric data, containing a wealth of relationships between various elements like rooms, walls, doors, and windows. A representation method that can effectively capture and convey these intricate geometric relationships is essential for accurate and meaningful floorplan generation. The chosen representation method influences how the problem is formulated and how solutions are generated. As shown in Table: 3, we introduce a classification system to categorize representations based on the subsequent desirable properties:

- **Handling geometry:** entails effectively encoding the shapes, sizes, positions, and orientations of architectural components to accurately represent the layout of a floorplan.
- **Capturing geometric relationships:** involves representing the spatial connections and arrangements between architectural elements such as rooms, walls, doors, and windows within a floorplan.
- **Loss of detail:** It is the extent to which fine-grained or specific information is not fully captured or represented in a particular representation or model.
- **Scalability:** It pertains to the representation's ability to efficiently accommodate and process floorplans of varying sizes and complexities, without sacrificing accuracy or computational efficiency.
- **Computational efficiency (Performance):** Efficient representations enable faster processing, which is especially important when dealing with large datasets or real-time applications.

In the context of the comparison Table 3, "High" signifies a high level of effectiveness and suitability for the property, "Moderate" indicates a reasonable but not extensive effectiveness, "Limited" points to a deficiency in addressing the property, and "Variable" conveys fluctuating effectiveness based on varying factors.

Pixel-based representation, though effective for image-based tasks, might fall short in encapsulating the nuanced spatial relationships within floorplans. While pixels can encode color and visual features, they might not inherently convey the precise positioning and connections that are integral to floorplan generation. Similarly, feature-based representation's [21, 22, 31, 42, 51] efficacy hinges on meticulous feature design that accurately captures geometric properties. This approach might not naturally lend itself to capturing complex architectural arrangements.

On the other hand, Vector-based representation [14, 20], meanwhile, holds promise due to its potential to encode geometric shapes and layouts efficiently. Vectors can represent lines, angles, and curves, allowing for the representation of intricate architectural components. Moreover, they can encode spatial relationships and dimensions, a crucial aspect of floorplan accuracy. Nevertheless, vector-based representation might demand careful handling to ensure the representation effectively captures the rich geometry of floorplan elements without sacrificing interpretability.

Graph-based representation [14–16, 19] emerges as a compelling choice due to its prowess in capturing complex relationships. Graphs excel at modeling connections between elements, directly aligning with the interplay between architectural components in a floorplan. Nodes can signify individual elements, while edges elegantly represent adjacency and spatial relationships. This structured

Table 3 A comparison of representation methods using desirable properties

Representation	Capturing Geometric Relationships	Handling Geometry	Loss of detail	Scalability	Performance
Pixel-Based [47]	Limited	Limited	Limited	Variable	High
Vector-Based [14, 20]	Moderate	High	Moderate	Variable	Moderate
Feature-Based [21, 22, 31, 42, 51]	Limited	Limited	Moderate	Moderate	Moderate
Graph-Based [14–16, 19, 23]	High	High	Limited	High	High

approach ensures that the generated floorplans maintain the intended spatial arrangement and proportions. However, graph-based methods might require careful interpretation, especially for complex layouts, and might pose scalability challenges for large-scale floorplans.

Pixel-based representations excel in retaining detailed spatial information and fine-grained features, resulting in a limited loss of detail. However, their high-dimensional input space demands significant computational resources. On the other hand, vector-based representations strike a balance between detail preservation and computational efficiency, though they may lose some fine-grained details while encoding numerical features. Feature-based representations capture high-level semantic information and domain-specific knowledge while being computationally efficient, but they require a feature extraction process that may lead to a moderate loss of low-level details. Graph-based representations minimize the loss of detail by capturing complex relationships, making them suitable for interconnected structured data and facilitating reasoning over graph structures. However, processing large graphs can impose moderate to high computational demands.

In conclusion, the choice of representation method for floorplan generation holds the key to creating accurate, intricate, and visually appealing layouts. While pixel-based and feature-based representations might not fully capture the structural intricacies of floorplans, graph-based and vector-based methods show promise in their ability to preserve spatial relationships, handle geometry, and capture the complex web of architectural connections. The decision ultimately hinges on the intricacy of the task, the level of accuracy required, and the trade-offs between representation complexity and interpretability.

Computational Methods

Computational-based methods for floorplan generation refer to the application of computational techniques, including algorithms, mathematical models, and data-driven approaches, to automatically create architectural layouts for buildings and interior spaces. These methods encompass a wide range of techniques, from rule-based systems that follow architectural guidelines to more advanced approaches utilizing machine learning to create innovative and aesthetically pleasing floorplans. By harnessing the power of computers and artificial intelligence, they analyze input parameters such as spatial requirements, user preferences, and design constraints to generate floorplans that optimize spatial organization, functionality, and aesthetics. These methods offer an efficient and innovative way to design floorplans, catering to various design objectives and scenarios across architecture and related fields.

Types of Computational Methods

As shown in Fig. 2, various types of computational methods are integral to the process of floorplan generation. These methods are grouped into procedural methods [3, 47] and deep learning methods [12, 15, 16, 19, 20, 52]. Procedural methods encompass algorithms like SubDivision [2], Tip Placement [4], Inside Out [5], and Growth-based algorithms [53], which rely on rule-based and algorithmic approaches to generate floorplans. Deep learning methods, including Diffusion Models [52], GANs [7], AutoRegressive models [9], and others, harness the power of artificial intelligence and neural networks to learn from existing floorplan data and generate innovative and aesthetically pleasing designs. This comprehensive range of computational methods empowers architects, designers, and urban planners to efficiently create and explore floorplan variations tailored to diverse design requirements and challenges.

Procedural Methods

It is a technique used to generate and create complex structures, scenes, or content automatically through the use of algorithms, rules, and parameters [54]. Instead of manually designing each element of the content, procedural modeling allows for the creation of detailed and varied assets by defining a set of rules and procedures that determine

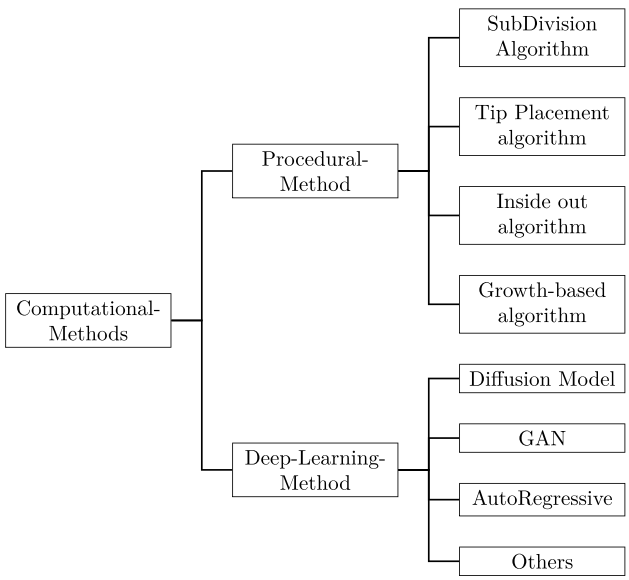

Fig. 2 Computational methods in floorplan generation

their appearance, arrangement, and behavior. While procedural content generation has a long history in gaming (e.g Elite [55]), recent efforts have started exploring large-scale, man-made 3D environments, particularly in city and building generation projects [56, 57]. In the realm of floorplan generation, some notable procedural techniques include Sub-Division [2], Tip Placement [4], Inside Out [5], and Growth-based algorithms [53], each offering unique approaches to creating architectural layouts.

Procedural techniques for floorplan generation vary across different areas, However, four types of procedural techniques are commonly used as the foundation for creating diverse and customizable floorplans.

A. Depth Peeling Algorithm

Depth peeling is an effective technique used in computer graphics for achieving order-independent transparency rendering [58]. This method involves rendering the image multiple times to accurately handle complex transparent objects and ensure correct depth sorting. With each iteration, the algorithm peels away the layers of transparent geometry, progressively capturing the depth information of each fragment. By iteratively blending the colors of the peeled layers, depth peeling resolves the visibility and occlusion issues associated with transparent objects, allowing for accurate and visually pleasing rendering of scenes that involve overlapping and translucent elements. This approach is particularly useful in various applications, such as architectural renderings [5, 6], medical imaging [59], and interactive simulations [60], where maintaining the correct visual hierarchy and transparency effects is crucial for realistic and immersive graphics.

B. Tile Placement Algorithm

Tile placement algorithms are computational methods employed to arrange tiles or tiles with specific properties in a manner that optimally fills a given space or adheres to predefined constraints [61]. These algorithms aim to efficiently position tiles while taking into account factors such as aesthetics, space utilization, and design requirements. By employing these algorithms, the goal is to achieve an optimal arrangement that maximizes the visual appeal, effectively utilizes the available space, and satisfies the desired design criteria. These techniques find applications in various domains, including interior design, urban planning, computer graphics [62], and game development, enabling the creation of visually pleasing and well-utilized tile layouts systematically and efficiently.

C. Subdivision Algorithm

It is a procedural technique that recursively divides space into smaller, more detailed components to generate complex structures by iteratively subdividing and refining basic shapes [2]. Subdivision algorithms are used in geometric modeling to generate smooth and detailed curves, surfaces, or structures.

D. Growth-Based Algorithm

A growth-based algorithm is a computational methodology used to model the emergence and development of complex structures, organisms, or systems by iteratively applying predefined rules and interactions [53]. These algorithms simulate natural growth processes, imitating how components evolve, interact, and organize themselves over time. Starting from an initial configuration, such as a seed or basic element, the algorithm progressively adds new components based on local interactions and specified growth rules. These rules dictate how new components connect, position themselves, and adapt to their environment. As the algorithm advances through iterations, the components' interactions lead to the emergence of intricate patterns and forms that resemble natural phenomena. Growth-based algorithms are applied in diverse domains, including 3D modelings [56], analysis [63], and medical modeling [64] to generate dynamic and visually appealing structures that mimic the behavior of living organisms or evolving systems.

Discussion

The discussion section on floorplan generation using procedural methods focuses on the utilization of computational algorithms to create floorplans based on predefined rules and constraints. Table 4, presents a summary of floorplan generation using different methods, outlining the following desirable properties achieved through this approach:

- **Type**: the type of procedural floor plan generation. The types are subdivision, tile placement, depth peeling, and room growth algorithms.
- **Window constraint:** determines whether the method uses windows as a constraint for room placement.
- **Space criteria:** This concerns whether user-provided requirements for a room's shape and placement in the floor plan are respected by the method.
- **Spatial Connectivity:** determines whether user-provided connection requirements (for example, the need for two rooms to be adjacent) are respected by the method.
- **Visualization:** It refers to the visual representation of the generated floor plans.

Table 4 Desirable properties for Procedural floorplan generations

Type	Name	Window constraint	Number of story	Space criteria	Spatial connectivity	Visualization
Depth peeling	Martin 2006 [5]	–	Single	✓	–	2D
	Merrell 2010 [6]	–	Multiple	✓	✓	2D and 3D
	Hahn 2006 [65]	–	Single	–	–	2D
Subdivision	Marson 2010 [66]	–	Single	✓	–	2D
	Rinde 2008[3]	✓	Single	✓	–	3d
Growth Based	Tutenel 2009 [67]	–	Single	–	✓	2D
	Lopes 2010 [68]	–	Single	–	✓	2D
Tile plac	Peng 2014 [4]	–	Single	–	–	2D

- **Space Utilization:** It focuses on optimizing the efficient use of available space within the floor plan.

The introduction of [5] presents the utilization of the procedural approach to generate customized floor plans specifically designed for residential buildings. This method comprises three key stages. During the initial stage, the fundamental layout of a house is portrayed through a graph, wherein nodes represent rooms and edges symbolize room connections. The graph is constructed with the front door as the root node, followed by the inclusion of public rooms, and subsequently private rooms. Specific room types are assigned after distributing public rooms, based on user-informed attributes. Progressing to the second stage, all rooms are assigned types and room connections are determined; however, the rooms are yet to be positioned spatially. In this phase, the algorithm calculates the 2D position of each room using a tree structure, treating the room adjoining the front door as the root and evenly distributing child nodes adjacent to it. This even distribution applies to both the spacing between child nodes and their distance from the root. The third stage encompasses room expansion to their final dimensions. Each room exerts an outward "pressure" proportional to its required size, leading to its expansion to fill the remaining space within the building. If two rooms share a wall, the algorithm assesses the pressures both inside and outside the wall to determine the extent of room expansion.

In their work, Merrell et al. [6] introduce a method for generating realistic residential building floor plans based on high-level user specifications. Recognizing the complexity and ambiguity of architectural rules, which prove challenging for conventional rule-based systems, the authors employ machine learning techniques. They develop a Bayesian network trained on real-world building data to infer subjective architectural design aspects that are difficult to explicitly define, such as room adjacencies, area, aspect ratio, and open versus door-adjoined adjacencies. The procedural generation process unfolds in two stages. Initially, a set of flexible high-level requirements, like bedroom count or approximate square footage, is defined. The Bayesian network then extends these requirements into a comprehensive architectural program, specifying room relationships and desired room attributes. Moving to the second stage, the architectural program is transformed into detailed floor plans for each building floor. This stage employs stochastic optimization within the space of potential building layouts. The algorithm iteratively generates new layouts, incorporating local and global reconfigurations that significantly alter the overall arrangement. These reconfigurations encompass sliding walls and room swaps, and the quality of each proposed layout is assessed by a cost function. This function considers factors such as accessibility, area, aspect ratio, and room shape, alongside a term penalizing irregular floor outlines. The combined method, involving machine learning-driven architectural inference and stochastic optimization, yields a practical approach to creating realistic and diverse residential building floor plans from user-specified high-level requirements.

The paper titled Persistent Realtime Building Interior Generation [65] introduces a novel approach for generating virtual building interiors in real-time. This method follows a top-down methodology guided by architectural guidelines, selectively generating only the necessary portions. By adopting a lazy generation scheme, it optimizes memory usage, utilizing significantly less memory compared to a complete interior model. This efficient memory management enables real-time frame rates, making it highly suitable for interactive applications. Moreover, the approach ensures that changes made within deleted regions are not lost, allowing for persistent modifications and creating a dynamic and consistent environment. This capability empowers developers to have greater control over the content by facilitating changes that persist beyond the lifespan of specific regions. To simplify implementation, the interior generation process

is divided into multiple stages, each marked by the assignment of a type to temporary regions. The stages encompass Building Setup, Floor Division, Hallway Division, Room Cluster Division, and Built Region Generation. Building Setup focuses on elements affecting multiple floors, including elevators, stairwells, and global aspects like textures. Floor Division uniformly divides the building into floors. Hallway Division constructs hallways around rectangular regions, while Room Cluster Division further subdivides the regions into rooms based on portal locations. The final stage, Built Region Generation, generates visible built regions by creating geometry within bounding box boundaries and placing objects accordingly. This staged approach streamlines the interior generation process, ensuring efficient memory utilization and facilitating the coherent and dynamic creation of virtual building interiors.

Another subdivision approach which generate house floor plans with semantic information is Automatic real-time generation of floor plans based on squarified treemaps algorithm [66]. The algorithm uses the squarified treemaps algorithm to divide the available space and connect the rooms together, placing doors between them based on a connection graph that is randomly produced at a previous stage based on some rules. The algorithm generates every aspect of the house randomly, resulting in dissimilar floor plans. The random generation includes the outer shape of the house, its area, number of rooms and their functionality, and the position of windows and doors. The algorithm generates the floor plan in a step-by-step approach, placing the rooms in the first level of the hierarchy, and then placing the smaller rooms below it inside. The algorithm provides real-time generation of floor plans, making it attractive for real-time interactive applications.

Rinde and Dahl [3] proposed to generate an indoor environment by using subdivsion method. The algorithm starts the process by taking the existing exteriors as input rather than creating the exterior by itself. The creation of a skeleton is another thing that needs to be considered and created by pushing all walls inwards at a constant rate and creating a skeleton edge where two walls meet. Now it's time to create a transition area, if the distance from the root node to closest wall is big enough, coridors are needed to minimize the number of rooms without windows. after the creation of corridors, the remaining space is split into maximum connected area, each with a continuous boundary the regions have been created, and they are further subdivided into sub-regions (in this case apartments). For apartments, it is important to make sure that each one has at least one window, windowless apartments being very rare. With sub-regions designed, the actual rooms within them should be created. When the room walls are created for an apartment, they are used to build the individual rooms. These are then connected to each other through entry points doors), and allocated a room type starting with a room which is connected to the transition area.

The papers authored by Tutenel et al. [67] introduce pioneering methodologies in interior generation and layout solving. Their approach for generating floor plans employs a growth-based method that utilizes a semantic library, representing each room type as a class with predefined relationships and constraints. Through iterative expansion of rooms and strategic placement of feature areas based on these relationships, the resulting floor plans adhere to specific constraints, resulting in well-structured and functional building layouts. Furthermore, their rule-based layout solving technique combines a semantic class library with a layout solver, enabling the generation of appropriate layouts for building floor plans and room arrangements. This comprehensive approach enhances various modeling techniques, including manual, automated, and mixed methods, leading to a more efficient layouting process for game worlds. The effectiveness of the approach is demonstrated through diverse layout problems addressed in the paper. Collectively, these papers provide valuable insights and practical solutions to advance interior generation and layout solving, facilitating the creation of impressive virtual building interiors and optimizing the layout generation process.

The paper titled "A constrained growth method for procedural floor plan generation" [68] introduces an algorithm that facilitates the growth of rooms based on a regular grid consisting of square cells. This grid structure imposes certain limitations on the growth process, as it aligns with the predetermined length of the cells and confines the expansion to the utilization of exterior walls exclusively. Consequently, the algorithm necessitates the placement of openings in the floor plan as the final step, in accordance with the procedural generation approach employed. Moreover, the algorithm incorporates a mechanism for controlling the expansion of rooms through the assignment of priority levels. By assigning higher priority values, rooms are granted a greater degree of expansion, resulting in proportional increases in size. For instance, a room with twice the priority of another will expand to twice its size. However, it is important to note that the algorithm lacks control over the final shape of the room and does not provide any guarantees regarding the attainment of a minimum size requirement. Despite these limitations, the proposed method offers a promising approach to procedural floor plan generation, enabling the creation of diverse layouts that adhere to user-defined constraints within the constraints of the grid-based framework.

Peng et al [4], present a method for tiling a domain with a set of deformable templates, such that the domain is completely covered and the templates do not overlap. The approach is suitable for a large class of applications like floorplans, urban layouts, and art and design. The method

involves a two-step process. The first step focuses on laying out the approximate positions of templates, while the second step refines the shapes of these templates. During the initial discrete stage, the layout algorithm tessellates both the domain and the templates into quadrilateral meshes. This tessellation breaks down the domain and template shapes into smaller quadrilateral elements, facilitating the subsequent layout and refinement processes. The next phase involves finding a tiling solution where the quadrangulated domain is seamlessly filled with copies of the quadrangulated templates. This gap-free tiling is achieved using an integer programming approach. However, since the placed tiles might not exactly match the original template shape due to deformations, the algorithm further refines the tiles. It employs a continuous quadratic optimization process that iteratively adjusts the placement of tiles until either a convergence criterion is met or a predefined time limit is reached. This optimization aims to improve the layout and alignment of tiles for better visual quality and coverage.

In their research, Martine [5], Merrel [6], Marson [66], and Rinde [3] focused on addressing space criteria, while also considering spatial connectivity. Merrel [5], Tutenel [67], and Lopes [68] also contributed to the discussion on spatial connectivity. However, Rinde's [3] specific focus was on addressing windows constraints. Overall, their work examines various aspects of addressing space and spatial constraints, providing valuable insights into the field.

Deep Generative Learning Method

While procedural methods have long been effective at generating structured and rule-based content, they often face limitations when it comes to capturing the intricate details and complexity found in real-world data. Procedural techniques rely on predefined algorithms and rules, which can result in repetitive and predictable outputs. This restricts their ability to accurately represent the rich and diverse nature of real-world environments. Moreover, procedural methods can be time-consuming and require significant manual effort to fine-tune parameters and achieve desired results [54]. They may struggle to achieve high levels of realism and variability, as they lack the ability to learn and generalize from large datasets. Procedural methods also face challenges in capturing the nuances of natural textures, lighting, and spatial relationships, resulting in outputs that may appear artificial or lacking in visual appeal. The evolution towards deep generative models has been driven by the need to overcome these limitations.

Generative Models have an extensive history in the field of artificial intelligence, dating back to the 1950s. Early models like Hidden Markov Models [69], Naive Bayes [70], and Gaussian Mixture Models [71] generated simple data. The development of these traditional generative models has impacted how contemporary generative models are designed and conceived, particularly within the framework of deep learning. A deep generative model belongs to a category of machine learning models that leverage deep neural networks to grasp and depict intricate data distributions. Unlike conventional discriminative models [72, 73], which prioritize predicting labels or making classifications, deep generative models strive to comprehend the fundamental structure of the data and generate new samples that resemble the original data. By using multiple layers of neural networks, these models can capture intricate patterns and dependencies present in the data. Deep generative models, such as Diffusion models [52], autoregressive model [9], Generative Adversarial Networks (GANs)[7], and others have demonstrated remarkable capabilities in tasks like image generation[33–36], text synthesis [37, 38], and so. They have significantly advanced the field of generative modeling and hold promise for various applications, ranging from creative art generation to scientific data synthesis [74, 75] and beyond.

In this section, our focus is on a specific category of generative models that have played a significant role in the generation of floorplans. While there are various types of generative models, we will delve into three types that have been particularly beneficial in the creation and design of floorplans: Generative Adversarial Networks (GANs) [7], autoregressive models [9], and diffusion models [52]. Each of these models brings unique contributions to the

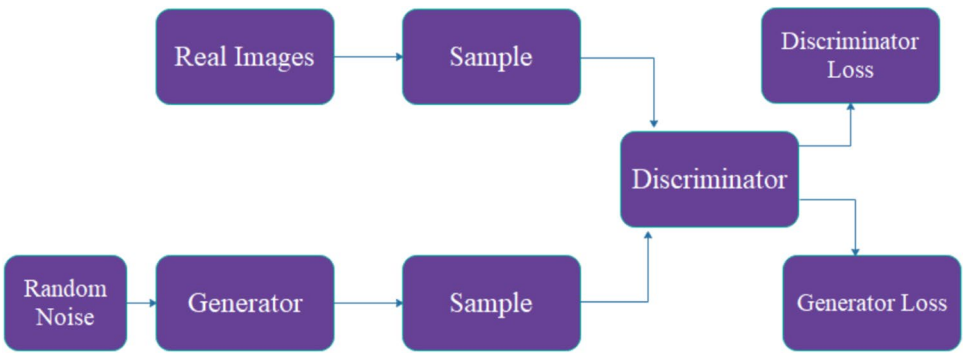

Fig. 3 Generative Adversarial Network

Table 5 Deep generative Model based floorplan Generation

Methods	Name	Diversity	Compatibility	Visual Realism	Functional Realism	Sampling Time
GAN	Wang et al. [12]	–	✓	–	✓	–
	Chaillou et al. [13]	–	✓	–	–	
	Nauata et al [15]	✓	✓	✓	–	–
	Nauata et al. [16]	✓	✓	✓	–	–
	Tang et al. [14]	✓	✓	✓	–	–
	Chailou et al. [18]	–	✓	✓	–	–
	Zheng et al. [31]	✓	✓	–	–	–
	Liu et al. [17]	–	✓	✓	–	–
	Chen et al. [42]	–	✓	–	–	
	Lim et al. [44]	✓	✓	–	–	–
	Schiller et al. [51]	–	✓	–	–	
Diffusion model	Shabani et al. [19]	✓	✓	✓	–	–
Autoregressive	Liu et al. [20]	✓	✓	✓	✓	–
Others	Liu et al. [21]	–	✓	✓	–	–
	Hu et al. [22]	✓	✓	✓	–	–
	Chen et al. [23]	–	✓	✓	–	–
	Wu et al. [90]	–	✓	✓	–	–

field, revolutionizing the way floorplans are generated and designed. We will explore how these models have advanced the process and brought something special to the domain of floorplan creation.

Generative Adversarial Network (GAN)

A Generative Adversarial Network (GAN) is a powerful and innovative deep-learning framework introduced by Ian Goodfellow and his colleagues in 2014 [7]. As shown in Fig. 3, GANs consist of two neural networks, the generator and the discriminator, which are trained in a competitive manner. The generator creates synthetic data samples, while the discriminator attempts to distinguish between real data and generated data. Through a process of iterative training, the generator improves its ability to produce realistic data, while the discriminator becomes more adept at differentiating real from fake data. Generative Adversarial Networks (GANs) have undergone significant evolution, resulting in various types and improvements that have revolutionized generative modeling. From Vanilla GANs' [7] adversarial setup to conditional GANs [76], DCGANs [77], PGANs [78], and WGANs [79]. Each of these variants brings unique advancements to the field. Additionally, InfoGANs [80] have contributed to promoting disentanglement in generated outputs. StyleGAN [81] and its variations have enabled realistic and controlled image synthesis by incorporating disentangled latent and hierarchical architectures. This evolution leads to the development of highly realistic synthetic data, making GANs widely used for tasks like image synthesis [82, 83], data augmentation [84], super-resolution[85], and style transfer [86, 87].

Mathematical Representation

Generative Adversarial Networks (GANs) can be described as an interplay between two distinct models: the generator and the discriminator. Let's denote the generator as G and the discriminator as D. The generator takes a noise vector z as input and generates fake samples G(z). The discriminator takes both real samples x and fake samples G(z) as input and outputs a probability score indicating the likelihood of the input being real or fake.

The generator loss, denoted as L_G, aims to minimize the divergence between the generated samples and the real samples. One commonly used loss function for the generator is the binary cross-entropy loss:

$$L\_G = -\log(D(G(z))) \tag{1}$$

The discriminator loss, denoted as L_D, aims to accurately classify real and fake samples. It involves two terms: the loss for real samples and the loss for fake samples. The discriminator tries to minimize this loss function while the generator tries to maximize it. By engaging in this adversarial training process, the discriminator and generator networks learn and improve their performance over time.

$$L\_D = -log(D(x)) - log(1 - D(G(z))) \tag{2}$$

Training of GAN

The training of Generative Adversarial Networks (GANs) involves optimizing the discriminator and generator networks through multiple training iterations (epochs). The assessment of its time complexity is framed in terms of the product of three key factors. Firstly, the number of training iterations (T) signifies the frequency of processing the entire dataset. Secondly, the dataset size (N) influences the computational load, with larger datasets demanding more time per epoch. Lastly, the complexity of the GAN's model architecture (f(model_architecture)), encompassing parameters, layers, and network intricacies, contributes significantly to the computation required.

$$O(T \cdot N \cdot f(\text{model\_architecture})) \tag{3}$$

The Eq. (3), succinctly captures the interplay of these factors in gauging the computational cost of GAN training, a critical consideration in the realm of generative models. The general intuition is that increasing T, N, or the complexity of the model architecture will generally lead to a higher time complexity for training the GAN.

Algorithm 1 Training a Generative Adversarial Network (GAN)

Require: Training dataset D, learning rate η, number of training iterations T, discriminator network $D(\cdot)$, generator network $G(\cdot)$
Ensure: Trained discriminator and generator parameters
Initialize discriminator and generator parameters θ_D and θ_G
for $t = 1$ to T **do**
  Sample a mini-batch from D
  Sample random noise $\mathbf{z}$ from a prior distribution
  Generate fake data samples $\mathbf{x}_{\text{fake}} \leftarrow G(\mathbf{z}; \theta_G)$
  Compute discriminator loss $L_D \leftarrow -\mathbb{E}_{\mathbf{x}\sim D}[\log D(\mathbf{x}; \theta_D)] - \mathbb{E}_{\mathbf{x}\sim G(\mathbf{z};\theta_G)}[\log(1 - D(\mathbf{x}; \theta_D))]$
  Update discriminator parameters $\theta_D \leftarrow \theta_D - \eta \cdot \nabla_{\theta_D} L_D$
  Sample new random noise $\mathbf{z}$ from a prior distribution
  Generate fake data samples $\mathbf{x}_{\text{fake}} \leftarrow G(\mathbf{z}; \theta_G)$
  Compute generator loss $L_G \leftarrow -\mathbb{E}_{\mathbf{x}\sim G(\mathbf{z};\theta_G)}[\log D(\mathbf{x}; \theta_D)]$
  Update generator parameters $\theta_G \leftarrow \theta_G - \eta \cdot \nabla_{\theta_G} L_G$
end for

The algorithm 1 involves alternating updates between the discriminator and generator networks over a specified number of training iterations (epochs). During each iteration, a mini-batch of real data is sampled, and the generator produces fake data samples from random noise. The discriminator and generator losses are computed based on their abilities to distinguish between real and fake samples. These losses guide parameter updates, where the discriminator aims to improve its ability to differentiate real from fake data, while the generator aims to produce more convincing data to fool the discriminator. This adversarial training process continues iteratively, resulting in trained discriminator and generator networks capable of generating data resembling the training dataset's distribution.

Sampling of GAN

Sampling of GAN refers to the process of generating new samples from the generator network, typically producing synthetic data that resembles the training data distribution. The time complexity for sampling from a Generative Adversarial Network (GAN) hinges upon the intricacies of the generator network's architecture and the dimensionality of the latent space (D). In practice, the computational effort required for generating samples is largely influenced by the complexity of the generator, encompassing factors such as the number of layers, parameters, and the computational operations involved in transforming latent noise into data samples. The latent space dimension (D) also

plays a pivotal role, as it determines the size of the space from which random vectors are drawn to generate data.

The time complexity can be succinctly expressed as:

$$O(f(\text{generator\_architecture}) \cdot D) \tag{4}$$

As shown in Eq. (4), the time required for sampling depends on the intricacy of the generator architecture and the number of samples to be generated. As the complexity of the generator architecture increases or the number of samples to be generated grows, the computational resources and time needed for sampling also increase. Therefore, careful consideration of the generator architecture and the desired number of samples is necessary to manage the time complexity of the sampling process effectively in GANs.

- **Method:** represents a specific approach or Model used to create a new floorplan.
- **Name:** highlights the paper name or used method with reference.
- **Diversity:** refers to the count of unique floorplans generated during a single sampling instance. It encompasses a range of distinct designs that adhere to specified constraints and layout arrangements.
- **Compatibility :** The ability of generated floorplans to seamlessly fit and adhere to the given design boundaries and spatial requirements.
- **Visual Realism:** The extent of similarity between generated designs and actual architectural layouts.

Algorithm 2 Sampling from a Generative Adversarial Network (GAN)

Require: Trained generator network $G(\cdot)$, latent noise $\mathbf{z}$, number of samples to generate N
Ensure: Generated samples $\mathbf{X}$
1: Initialize empty set $\mathbf{X}$
2: **for** $n = 1$ to N **do**
3: Sample random noise $\mathbf{z}$ from a prior distribution
4: Generate a sample $\mathbf{x}_{\text{sampled}} \leftarrow G(\mathbf{z})$
5: Add $\mathbf{x}_{\text{sampled}}$ to $\mathbf{X}$
6: **end for**

The above algorithm 2 outlines the process of sampling from a trained Generative Adversarial Network (GAN), a core element in generative modeling. By leveraging a trained generator network, this algorithm generates synthetic data samples from random latent noise vectors. During each iteration, a random noise vector is sampled from a prior distribution and passed through the generator network to produce a synthetic sample. The generated samples are then accumulated into a collection.

Discussion

Previous research into the utilization of Generative Adversarial Networks (GANs) within architectural design spans a wide array of domains. Examples of these investigations include transforming city maps into satellite imagery [88], generating furniture layouts [89], and urban planning [27]. Furthermore, GANs have found more recent applications in simplifying layout design processes, such as generating layouts for residential buildings.

Table 5, presents a summary of floorplan generation using different generative models, outlining various desirable properties achieved through this approach.

- **Functional Realism:** The extent to which generated floorplans accurately represent functional and logical relationships between rooms and spaces, ensuring practicality and usability.
- **Sampling Efficiency:** The effectiveness of the floorplan generation process in exploring diverse design possibilities while using minimal computational resources and iterations.

And also the "✓" symbol signifies that the papers have addressed the specified constraint, while the "-" symbol indicates that the constraint was not taken into consideration by those papers. From the analysis, it appears that a significant number of researchers have focused on addressing compatibility and enhancing visual realism in floorplan generation, with some attention given to diversity. However, the functional realism of floorplans has often been overlooked. Another aspect that has not received much consideration is the sampling efficiency of floorplan generation, particularly its suitability for real-time applications.

To harness the advantages of GAN, different researchers use GAN at different times to generate various elements in architectural designs, including the generation of furniture Arangments [91], the recognition of different rooms,

the transformation of city maps [92], and volumetric design generation [93].

The findings presented in the Table 5 reveal a discernible pattern among researchers, particularly those denoted as [14–16, 44] and others who have demonstrated a pronounced capability to generate diverse images. Additionally, the majority of generative models rooted in GAN architecture exhibit a commendable ability to create compatible floorplans. Another noteworthy consideration involves the facets of visual realism and functional realism. Researchers [14–18] stand out for their proficiency in generating visually captivating floorplans; however, an omission in their approach pertains to the oversight of functional realism—an aspect which ensures that the generated floorplans are not just aesthetically pleasing, but also practically viable.

The paper presented by [31] for generating floorplans, aiming to create a tool that transforms input images featuring design boundaries into detailed interior designs within those boundaries. The method involves training on two distinct datasets and includes preprocessing steps such as boundary production and masking. The model's training employs Pix2pixHD on labeled datasets comprising resized images and their corresponding masked versions. While the model tackles compatibility challenges, it faces drawbacks such as extended training times and difficulty in accurately placing key spaces like living rooms, kitchens, and bedrooms. Moreover, issues related to achieving visual and functional realism and customization remain to be addressed.

The paper presented by [12] for generating floorplans, aims to generate diverse floorplans for residential buildings that meet the conditions of human-environment interaction outlined in the activity map. Unlike other methods, they use a human activity map which is extracted from the input boundary and used to guide the floorplan generations from the input boundaries. This human activity maps is performed either automatically with a GAN model trained from synthetic human-activity maps or semi-automatic approach by using bi-RRT [94] based on user-specified furniture locations. To produce the vectorized floorplans the paper proposes two-stage approaches, the first stage is named as ActFloorr-GAN and aims to synthesize a rasterized human activity map from the input boundary. Then this pixel-wise representation is converted into a vectorized way in the second stage. While they didnt provide the exact measurement, they tried to address functional realism by using a human activity map. They generate diverse floorplans but they didn't show is it possible to generate more than one floorplans. In extracting the Human activity map the modis not optimal. Uss DCGAN model.

A new approach introduced by [15] is HouseGAN by involving a generative adversarial network with graph constraints, where both the generator and discriminator utilize a relational architecture. The central concept is to incorporate the constraint within the relational network's graph structure. To achieve this, the model employs conv-MPN (Convolutional Message Passing Network) [95] for graph updates and upsampling in the generator, as well as downsampling in the discriminator. However, it's essential to recognize the approach's limitations, including restricted rectangular shape generation, the absence of room size incorporation, and the omission of door placements in the graph's edges due to spatial adjacency, all of which suggest potential avenues for future refinements and expansions of the proposed method.

In order to solve the problem of houseGAN [15, 16] provides an updated version by combining a relational GAN constrained by graphs and a conditional GAN. This integration allows for iterative improvement, as a previously generated layout serves as the next input constraint. Notably, this research unveils the effectiveness of a simple non-iterative training approach known as component-wise GT-conditioning in training such a generator. Moreover, the iterative generator presents a new avenue for refining chosen metrics through meta-optimization strategies by regulating the timing of input constraint passage during the iterative layout enhancement process.

Furthering the analysis, researcher [12] method stands out as it employs trace movement, effectively striding towards achieving functional realism by accounting for the movement dynamics within the floorplan's layout. This particular emphasis on functionality adds a layer of practicality to the generated designs. The discourse also shifts towards computational resources, an aspect that many researchers regrettably do not explicitly address. Nevertheless, through an inference drawn from the methods employed, it becomes conceivable to speculate about potential computational constraints. Notably, researcher [14] methodology emerges as a standout, showcasing superior resource efficiency in comparison to its counterparts. In conclusion, the implications of this exploration underscore a blend of capabilities, spanning

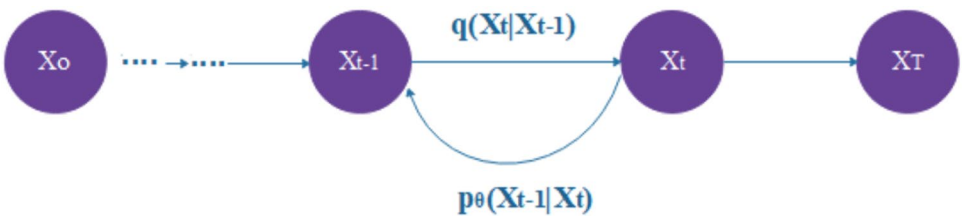

Fig. 4 Forward and backward Diffusion Process

diversity, compatibility, visual and functional realism, and judicious utilization of computational resources across various approaches in the field of generative floorplan design.

Generative Diffusion Model

The diffusion generative model is a type of generative model that operates by iteratively transforming a random noise signal to generate high-quality samples. It leverages the concept of diffusion processes, which involve gradually spreading or diffusing information over time [52]. In the context of generative modeling, diffusion models learn a sequence of transformations that progressively refine the initial noise signal, leading to the generation of realistic samples. These models are trained by optimizing the parameters to minimize the difference between the generated samples and the target data distribution. Diffusion generative models have gained attention for their ability to generate diverse and high-fidelity samples, and they have found applications in various domains, including image synthesis [96], style transfer [97], text [98], and audio [99]. They offer a promising approach to generative modeling by utilizing diffusion processes to capture complex dependencies and generate realistic and coherent samples.

Mathematical Representation

As shown in Fig. 4, the diffusion model comprises two processes Forward and Backward, where the forward process involves generating noise through a fixed noise vector, containing random values or samples from basic distributions like Gaussian noise, with the vector's size aligned with the size of the intended generated data, such as images or text sequences. The noise vector is passed through a sequence of diffusion steps.

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t;\mu_t = (1-\beta_t)x_{t-1}, \Sigma_t = \beta_t I) \tag{5}$$

The Eq. (5) above embodies the forward diffusion process within a probabilistic framework. This equation characterizes how a random variable x evolves over discrete time steps, with x_t representing its state at time t. The conditional distribution $q(x_t|x_{t-1})$ captures the likelihood of x_t given the previous value x_{t-1}, and is modeled as a Gaussian distribution $\mathcal{N}$. The mean μ_t of this distribution is determined by $(1-\beta_t)x_{t-1}$, reflecting how x_t depends on its prior state x_{t-1} with the influence controlled by the parameter β_t. Additionally, the covariance matrix Σ_t is specified as β_t times the identity matrix I, regulating the variability of x_t and indicating the level of uncertainty. Noise scheduling [100] is an important aspect of diffusion models that involves adding the right amount of noise to arrive at an isotropic Gaussian distribution with various types of schedules like linear [52], cosine [100], or combined approaches that determine how the noise increases over time.

Another integral component of the diffusion model is the Reverse Process, which involves a sequence of iterative steps that typically extend over hundreds to thousands of iterations. During each iteration of the reverse process, the noise vector undergoes sequential updates aimed at refining its representation. This refinement is accomplished through conditioning based on the actual training data. By progressively applying these iterative updates, the model learns to align the noise vector with the underlying patterns and structures present in the training data.

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1};\mu_\theta(x_t,t), \Sigma_\theta(x_t,t)) \tag{6}$$

In the backward diffusion process (Eq. 6), the model iteratively updates the noise vector by conditioning it on the actual training data. In the conditional distribution where x_t represents the current value in the diffusion process, and x_{t-1} represents the previous value. This conditional distribution is assumed to follow a multivariate normal distribution, with mean $\mu_\theta(x_t,t)$ and covariance $\Sigma_\theta(x_t,t)$. These mean and covariance parameters are learned during the training of the diffusion model. To refine the noise vector in the backward diffusion process, the model samples from the conditional distribution. This sampling step allows the noise vector to progressively align with the target data distribution, capturing the complex patterns and dependencies present in the training data. By iteratively applying the backward diffusion process, the model updates the noise vector over a series of steps, often spanning from hundreds to thousands. This sequential refinement facilitates the generation of high-quality samples that closely resemble the training data. The backward diffusion process, along with the forward diffusion process, enables the diffusion generative model to generate diverse and realistic samples by leveraging the learned conditional distribution and the associated mean and covariance parameters.

Training of Diffusion Model

Training diffusion models entails an iterative process wherein data is denoised and reconstructed to learn the underlying probability distribution. The worst-case time complexity of training diffusion models depends on multiple factors, including the number of denoising steps (S), dataset size (N), model architecture complexity, and the number of training iterations (T). As the number of denoising steps increases, so does the complexity, with each step requiring the processing of the entire dataset. A larger dataset demands more computational resources, and the model's architectural intricacy and the number of training iterations also impact the overall training time. As shown in Eq. (7),

the worst-case time complexity often reflects the interplay of these factors in determining the computational cost of training diffusion models.

$$O(T * S * N * f(model\_architecture)) \tag{7}$$

The algorithm 3 trains a diffusion model by adjusting its parameters based on a given dataset. It shuffles and divides the dataset into smaller parts, called mini-batches. Within each mini-batch, the algorithm performs denoising steps to improve the generated samples. It uses random noise sampled from a standard normal distribution to create new samples by applying the model architecture. The quality of the generated samples is assessed using a denoising loss function. The model parameters are then updated using gradient descent, with the learning rate determining the step size. By repeating this process for a specified number of iterations, the diffusion model learns to generate high-quality samples that resemble the training dataset.

Sampling of Diffusion Model

The sampling from a diffusion model is the process of generating new data points that follow the learned distribution. The time complexity depends on the number of diffusion steps (S), the model architecture, and the sequence length (L).

$$O(S * f(model\_architecture) * L) \tag{8}$$

The provided algorithm 4 outlines the process of sampling from a diffusion model, a generative model used in deep learning. This algorithm starts with an initial data point and iteratively adds noise to it, then gradually removes the noise to produce a new data point. The number of denoising steps, determined by the hyperparameter, influences the complexity of the sampling process. At each step, the noise is sampled from a standard Gaussian distribution. The model architecture, represented by the function f(model_architecture), guides the denoising and controls how the noise is incorporated. After all the denoising steps are completed, the final data point represents a sample generated from the diffusion model, capturing the underlying data distribution's characteristics. This algorithm is crucial for various generative tasks and data generation applications.

Algorithm 3 Training a Diffusion Model

Require: Training dataset D, number of denoising steps S, learning rate η, model architecture $f(\text{model\_architecture})$, number of training iterations T

Initialize model parameters θ
for $t = 1$ to T **do**
  Shuffle and split D into mini-batches
  for each mini-batch B in D **do**
    for $s = 1$ to S **do**
      Sample noise $\epsilon_s \sim \mathcal{N}(0, I)$
      Generate samples $x_t^s \leftarrow f(\text{model\_architecture})(\theta, x_t^{s-1}, \epsilon_s)$
    end for
    Compute denoising loss $L_t \leftarrow \text{DenoisingLoss}(x_t^S, B)$
    Update model parameters $\theta \leftarrow \theta - \eta \cdot \nabla L_t$
  end for
end for

Algorithm 4 Sampling from a Diffusion Model

Require: Diffusion model parameters θ, number of diffusion steps S, initial data point x_0
Ensure: Sampled data point x_{sampled}

Initialize $x_t \leftarrow x_0$
for $s = 1$ to S **do**
  Sample noise $\epsilon_s \sim \mathcal{N}(0, I)$
  $x_t \leftarrow f(\text{model\_architecture})(\theta, x_t, \epsilon_s)$
end for
$x_{\text{sampled}} \leftarrow x_t$

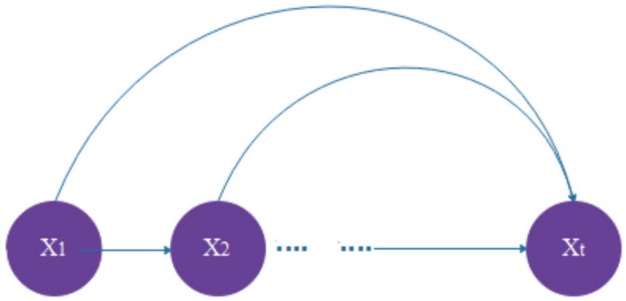

Fig. 5 Auto regresive Process

Discussion

The incorporation of diffusion models spans various layout generation applications, including document layout generation [8, 101], where diffusion models organized the arrangement of document elements to shape comprehensive layouts. Demonstrating this versatility, the house-Diffusion generative model [19], as indicated in Table 5, endeavors to generate vectorized floorplans seamlessly using diffusion processes. This paper introduces a groundbreaking approach that leverages a diffusion model and a core Transformer architecture [102] for the generation of intricate vector-based floorplans. These floorplans, comprised of interconnected polygons outlining rooms and doors, are created through a process guided by attention masks based on graph-constraints. This process involves a combined discrete and continuous noise reduction approach, resulting in accurate geometric relationships among architectural elements.

The model's direct generation of vector-based representations, facilitated by a Diffusion Model and enhanced by a Transformer network module, ensures the refinement of 2D pixel coordinates in both discrete and continuous forms. Notably, this approach integrates three attention mechanisms within the Transformer module, leveraging the structural connections among architectural components. The paper substantiates its claims through qualitative and quantitative evaluations, exhibiting remarkable advancements over prevailing methods across diverse metrics. A standout achievement is the model's ability to generate non-Manhattan structures and regulate corner counts with precision. As an innovative approach, this method introduces a direct generation of structured vector-graphic geometries for floorplans. However, it is important to note that the method does not specifically address the issues of sampling time and functional realism. These aspects may require further exploration and consideration, especially when dealing with larger-scale buildings.

Autoregressive Model

Autoregressive models are a class of generative models that capture dependencies within sequential or structured data. As shown in Fig. 5, the model is designed to generate new samples by estimating the conditional probability of each element in the sequence given the previous elements. By iteratively generating data elements, autoregressive models excel at capturing intricate patterns, making them especially effective in scenarios where the order and context of elements matter significantly. Autoregressive models have shown significant success in various domains, including natural language processing [103, 104] and computer vision [105, 106]. Notably, autoregressive models have witnessed substantial advancements, with innovative architectures like Transformer-based models [102] achieving exceptional performance in language understanding and generation tasks.

Mathematical Representation

$$p(x) = \prod_{i=1}^{n} p(x_i|x_1, x_2, \ldots, x_{i-1}) = \prod_{i=1}^{n} p(x_i|x_{<i}) \tag{9}$$

The Eq. (9), above encapsulates the foundational concept of an autoregressive model, widely employed across various generative tasks. In this context, x denotes a sequence of elements, often corresponding to pixels within images, while x_i signifies the ith element within the sequence. The equation's essence lies in expressing the probability distribution of the entire sequence x as a product of conditional probabilities. Each element's likelihood $p(x_i|x_{<i})$ is intricately modeled with respect to all prior elements, thus capturing complex dependencies existing within the sequence. This formulation allows autoregressive models to generate new samples by iteratively predicting each element in the sequence, resulting in the generation of outputs that exhibit complex patterns and closely resemble the characteristics of the training data.

Training of Autoregressive Model

Training autoregressive models is a fundamental process in deep learning where a model learns to generate sequences of data. The primary objective is to capture the underlying probability distribution of the sequences in the training data. During training, the model is exposed to input sequences one step at a time, and it sequentially predicts each element in the sequence based on previously generated elements. The training process involves minimizing a loss function that quantifies the difference between the model's predictions and the actual sequence data. Autoregressive models often use

techniques like teacher forcing, where the model is provided with ground-truth data during training, and autoregressive sampling during inference, where it generates sequences step by step. As shown in Eq. (10) the complexity of training depends on the sequence length (L), the number of model parameters (P), and the number of training iterations (T).

$$O(T * L * P) \tag{10}$$

The training algorithm for autoregressive models, as depicted in Algorithm 5, follows an iterative optimization approach to update the model parameters by minimizing prediction errors over sequential data. The algorithm processes a training dataset, splits it into mini-batches, and predicts each element in a sequence while updating hidden states iteratively. The loss incurred at each prediction step is accumulated to compute the batch loss, and model parameters are adjusted using backpropagation through time (BPTT). Through a series of training iterations, this process enables autoregressive models to capture sequential dependencies and generate coherent sequences, making it fundamental for tasks like natural language processing and time series forecasting.

Algorithm 5 Training an Autoregressive Model

Require: Training dataset D, model architecture $f(\text{model\_architecture})$, learning rate η, number of training iterations T
Ensure: Trained model parameters θ
Initialize model parameters θ
for $t = 1$ to T **do**
  Shuffle and split D into mini-batches
  for each mini-batch B in D **do**
    Initialize hidden state h_0
    for $i = 1$ to sequence length L **do**
      Input x_i from B and previous hidden state h_{i-1}
      Compute model prediction
      $y_i \leftarrow f(\text{model\_architecture})(\theta, x_i, h_{i-1})$
      Compute loss $L_i \leftarrow \text{Loss}(y_i, x_{i+1})$
      Update hidden state h_i
    end for
    Compute batch loss $L_B \leftarrow \sum_{i=1}^{L} L_i$
    Update model parameters $\theta \leftarrow \theta - \eta \cdot \nabla L_B$
  end for
end for

Sampling of Autoregressive Model

In autoregressive models, the process of sampling unfolds sequentially, with each element of a sequence generated based on the preceding ones. The worst-case time complexity for this sampling procedure depends on two key factors: the desired sequence length (L) and the intricacies of the model architecture (f(model_architecture)). Longer sequences naturally require more computation, as each element must be generated in sequence, while the complexity of the model architecture impacts the efficiency of each generation step.

$$O(f(\text{model\_architecture}) \cdot L) \tag{11}$$

The Eq. (11), signifying that the computational cost of sampling from autoregressive models scales with both the sequence length and the intricacy of the underlying model architecture.

Algorithm 6 Sampling from an Autoregressive Model

Require: Trained autoregressive model parameters θ, initial context x_0, sequence length L
Ensure: Sampled sequence $\mathbf{X}$
Initialize sampled sequence
$\mathbf{X} \leftarrow [x_0]$
for $i = 1$ to $L - 1$ **do**
    Compute next element x_i using autoregressive model
    $x_i \leftarrow f(\text{model\_architecture})(\theta, \mathbf{X}_{1:i})$
    Append x_i to $\mathbf{X}$
end for

The algorithm 6 outlines the process of sampling from an autoregressive model, a fundamental approach in sequential data generation and prediction tasks. Beginning with an initial context, it sequentially generates a new element for each position in the desired sequence length. This generation process relies on the autoregressive model's learned parameters and architecture, which conditions each prediction on the preceding elements in the sequence. The resulting sampled sequence encapsulates the model's understanding of sequential dependencies and serves as a valuable tool for tasks like text generation, time series forecasting, and various sequential data applications.

Discussion

The paper [20] introduces a novel autoregressive approach to synthesizing floorplans using 1-D vector sequences, enhancing user interaction and customization. The framework consists of a two-stage process involving a draft stage and a multi-round refining stage. The initial floorplan sequence is generated using a graph convolutional network (GCN) and an autoregressive transformer network. A panoptic refinement network (PRN) refines the design in the second stage, aided by a geometric loss to ensure proper room connectivity. As shown in 5, in contrast to prior methods this vectorized approach produces more realistic and functional designs, achieving higher usability and visual appeal by using panoptic refinement network (PRN). The framework's effectiveness is demonstrated through experiments on real-world floorplan data, showcasing its superiority over previous state-of-the-art methods.

Table 6 Cross over comparisons

Attributes	GAN	Diffusion Model	Autoregressive Model
Realism	★ ★ ★ ★ ★	★ ★ ★ ★ ★	★ ★ ★★
Diversity	★ ★ ★	★ ★ ★ ★ ★	★ ★ ★★
Sampling-efficiency	★ ★ ★★	★ ★ ★	★★

Others

In contrast to the previously mentioned generative models such as GANs, diffusion models, and autoregressive models, this approach delves into the realm of floorplan generation using distinct techniques like Convolutional Neural Networks (CNNs) [49], Graph convolutional Networks (GCNs) [107] or other techniques. By leveraging CNNs, the model harnesses spatial hierarchies and patterns to create floorplans that adhere to architectural constraints and aesthetic considerations. In parallel, GCNs enable the incorporation of spatial relationships and connectivity in the design, ensuring coherent room layouts and functional arrangements. This departure from traditional methodologies showcases an innovative direction in floorplan generation, emphasizing the power of CNNs and GNNs in capturing spatial intricacies and offering novel avenues for creating intelligent and user-centric architectural layouts.

Discussion

Prior to the utilization of conventional generative models [7, 9, 19] in the domain of floorplan generation, researchers employed a variety of deep learning techniques to explore innovative approaches. As shown in Table 5, in 2018 [21] proposes a Unified Framework for Floorplan Reconstruction from 3D Scans. The research focuses on automating indoor floorplan reconstruction by utilizing a smartphone's RGBD streams captured while walking through a house. The proposed solution, FloorNet, introduces a unique deep neural architecture that effectively addresses the challenge of processing vast 3D space data. FloorNet employs three

Table 7 Comparison of procedural methods and deep generative models

Proporties	Procedural methods	Deep generative models
Capturing intricate details and complexity	Often struggle to capture the intricate details and complexity found in real-world data	Excel at capturing intricate details and complexity through learning from large datasets
Time consumption	May require significant time and manual effort to fine-tune parameters and achieve desired results	Training phase can be time-consuming, but once trained, generation is typically faster
Realism and variability	Limited realism and variability compared to real-world data	Offer greater realism and variability by learning from large datasets
Capturing nuances of natural textures	May face challenges in capturing the nuances of natural textures, resulting in outputs appearing artificial	Can capture the nuances of natural textures, resulting in more realistic and visually appealing outputs
Visual appeal	Lack visual appeal due to limited realism and inability to capture fine details	Generate visually appealing outputs by capturing intricate details
Energy consumption	–	Can potentially achieve energy efficiency benefits compared to procedural methods

neural network branches: PointNet for 3D point processing, CNN with 2D point density images for enhanced local spatial reasoning, and CNN with RGB images to utilize full image information. These branches exchange intermediate features to harness the strengths of all architectures. A benchmark was established using RGBD video streams from 155 residential spaces, demonstrating FloorNet's efficacy in improving reconstruction quality through both qualitative and quantitative evaluations.

Another groundbreaking study [22] introduces an automated approach to floorplan generation that fuses deep neural networks with user-guided design.Their framework, Graph2Plan, employs a layout graph and user-defined constraints to produce floorplans that adhere to layout and boundary requirements. By allowing users to input room counts and constraints, the system retrieves floorplans from a database and uses Graph2Plan to convert layout graphs into refined room representations. The neural network, trained on a sizable annotated dataset, employs graph neural networks and conventional image convolution to process layout graphs, building boundaries, and raster floorplan images. The method's versatility and quality are demonstrated through its ability to accommodate diverse user inputs.

Addressing the intricate challenge of generating 3D house models based on linguistic descriptions, [23] introduces a House Plan Generative Model (HPGM) that uniquely divides the process into two sub-tasks: constructing layouts and synthesizing textures. To effectively tackle these tasks, two specialized modules, the Graph Conditioned Layout Prediction Network (GC-LPN) and the Language Conditioned Texture GAN (LCT-GAN), are proposed. These modules focus on generating floor plans and corresponding interior textures guided by provided descriptions. The generation of building layouts that fulfill the specified requirements is facilitated by the Graph Conditioned Layout Prediction Network (GC-LPN), which integrates adjacent information into the extracted features using a Graph Convolutional Network (GCN) [107], thereby enhancing the performance of layout generation.

The predominant focus of existing research revolves around addressing issues of compatibility and visual realism in the context of floorplan generation. However, both [21] and [23] encounter challenges when attempting to generate diverse floorplans within a single sampling, thereby compromising diversity. Furthermore, the mechanisms by which these models efficiently conduct sampling remain largely unaddressed. Another significant gap in the literature pertains to the issue of functional realism, a dimension that most papers fail to adequately consider or explore in their floorplan generation methodologies.

Comparative Analysis of Generative Learning Approaches with Cross-Over

As shown in Table 6, we provide a Comparative analysis of Generative Learning Approaches for floorplan generation based on sampling efficiency, realism, and diversity. The assessment is presented in a tabular format, featuring star rankings ranging from 1 to 5. The star rankings are categorized as follows: 5 stars represent "High," 4 stars indicate "Moderate," 3 stars signify "Medium," and, 2 stars reflect "Low," This systematic comparison enables a comprehensive understanding of the relative strengths and weaknesses of different approaches within the realm of Generative Learning for floorplan generation.

Realism: The underlying methodologies of generative models such as GANs, Diffusion models, and autoregressive models shape the realism achieved in floorplan generation. GANs [7], through a competitive training process between a generator and discriminator, generate highly realistic images and can extend this capability to floor plan generation by

learning and analyzing patterns, spatial relationships, room adjacencies, and circulation patterns from a dataset, enabling the generation of realistic layouts. Conversely, diffusion models, with their iterative refinement process and focus on data distribution modeling, tend to generate more realistic floorplans by progressively capturing intricate spatial relationships and details [19]. Autoregressive models, while capable of producing coherent and structured outputs, might encounter challenges in capturing global context and spatial coherence due to their sequential generation process [108].

Diversity: In the context of generating floorplans, the diversity of designs produced by Generative Adversarial Networks (GANs), Diffusion Models, and Autoregressive Models varies due to their distinct mechanisms. GANs engage in a generator-discriminator interplay, prioritizing realistic samples that mimic training data, which can lead to limited diversity due to mode collapse [7]. Conversely, Diffusion Models, operating through iterative diffusion processes, inherently explore a broader range of data variations, yielding more diverse floorplans [52]. Autoregressive Models generate diversity based on a sequential placement of components, but they might struggle to capture global layout variations, leading to moderate diversity [108].

Sampling Efficiency: Generative Adversarial Networks (GANs) are often perceived to have better sampling efficiency compared to diffusion models and autoregressive models in certain contexts. GANs employ a generator-discriminator interplay during training, allowing them to learn the data distribution more directly and potentially yield faster convergence [109]. However, diffusion models and autoregressive models can experience slower sampling efficiency due to their inherent mechanisms. Diffusion models require multiple iterative steps to refine a sample, which can become computationally intensive and slow down the generation process [110]. Similarly, The sampling process of autoregressive models is sequential in nature and typically scales linearly with respect to the data dimension [108] This means that the generation of each data point depends on the previous points, which can slow down the process.

Comparison of Procedural Methods and Deep Generative Models

The comparison between procedural methods and deep generative models as shown in the Table 7, highlights several important factors for floorpan generation. Procedural methods [54] excel at generating structured and rule-based content but often struggle to capture the intricate details and complexity found in real-world data. This limitation arises due to the reliance on predefined algorithms and rules, resulting in outputs that may appear repetitive and predictable [3]. Procedural methods also face challenges in achieving high levels of realism and variability, as they lack the ability to learn and generalize from large datasets [54]. Additionally, capturing the nuances of natural textures and achieving visual appeal can be difficult for procedural methods, leading to artificial-looking outputs. These limitations can hinder their suitability in domains where realism, variability, and fine-grained details are crucial.

In contrast, deep generative models offer significant advantages in addressing the shortcomings of procedural methods. By leveraging powerful machine learning techniques, deep generative models can learn from extensive datasets, allowing them to capture intricate details and complexity in content generation. This ability to learn from data results in outputs that are more realistic, visually appealing, and diverse compared to procedural methods [90]. Deep generative models excel at capturing the nuances of natural textures, enabling them to generate outputs that closely resemble real-world counterparts [11]. Moreover, the training phase of deep generative models can be time-consuming, but once trained, the generation process is typically faster [7]. The ability of deep generative models to capture intricate details, achieve realism, and generate visually appealing outputs has led to a noticeable transition from procedural methods to deep generative learning in floorplan generation.

The Importance of Generating Floorplans Using Computational Method

Generating floorplans using computational method can offer several important advantages:

- **Speed and Efficiency**
  Generative models revolutionize the architectural design process by vastly enhancing speed and efficiency. Unlike traditional methods, which often demand substantial time and labor investments, generative models automate the generation of floorplans, producing multiple designs swiftly and with minimal human intervention. This automation not only accelerates the design phase but also conserves valuable resources. By expediting the creation of floorplans, architects and designers can allocate more time to refining designs and exploring innovative solutions, ultimately streamlining the entire design process and enhancing productivity.
- **Variety and Creativity**
  Generative models herald a transformative shift in architectural design by offering architects and designers an expansive canvas to explore uncharted territories of creativity. Through their capacity to generate diverse floorplan designs, these models empower professionals to transcend conventional boundaries and delve into layouts

and configurations previously unexplored. By freeing designers from the constraints of traditional methodologies, generative models unlock a realm of possibilities, fostering an environment ripe for innovation. This flexibility not only broadens the scope of architectural imagination but also encourages the discovery of unconventional yet highly functional design solutions. Consequently, architects can push the boundaries of creativity, resulting in more inventive and original designs that cater to the evolving needs and aspirations of inhabitants.

- **Customization and Personalization**

  Computational methods represent a pivotal advancement in architectural design, particularly in the realm of residential spaces, where individual preferences and specific requirements reign supreme. By training these models on predefined design preferences or constraints, architects can harness the power of customization to craft floorplans tailored precisely to the needs and desires of homeowners. Whether it's optimizing for open spaces to accommodate social gatherings or prioritizing privacy with secluded areas, generative models offer the flexibility to translate abstract ideas into tangible floorplans. This level of customization not only enhances the functionality and comfort of living spaces but also fosters a deeper sense of personal connection and satisfaction for homeowners, ensuring that their homes truly reflect their unique lifestyles and aspirations.

- **Sustainablity and energy efficiency**

  The utilization of computational methods holds immense promise in advancing sustainable development goals, particularly within the framework of the United Nations' Sustainable Development Goals (SDGs), notably SDG 11 (Sustainable Cities [111] and Communities) and SDG 13 (Climate Action) [112]. A compelling case study emerges from the Municipality of Athens, showcasing the transformative potential of AI-driven approaches in energy management, climate modeling, and sustainable energy optimization. Through harnessing the capabilities of Generative models, Athens serves as a beacon of technological innovation, illustrating how advancements in AI can catalyze a shift towards greener, more resilient urban environments. This convergence of cutting-edge technology, environmental consciousness, and strategic policymaking propels Athens to the forefront of sustainable urban development, demonstrating a holistic approach to environmental stewardship and community well-being. In this context, the significance of AI-based floorplan generation lies in its transformative impact on the architectural industry's ability to create environmentally responsible buildings. By harnessing AI algorithms, architects can optimize resource utilization, enhance energy efficiency, and improve indoor environmental quality from the earliest stages of the design process. This technology enables the generation of floorplans that prioritize sustainability metrics such as natural lighting, energy consumption, and material usage, leading to the creation of buildings that are both environmentally conscious and economically viable. Furthermore, AI-driven simulations and predictive analytics empower architects to evaluate the environmental performance of designs in real-time, facilitating informed decision-making and iterative improvements throughout the design process. Ultimately, this computational-based floorplan generation not only accelerates the adoption of sustainable design practices but also contributes to the development of healthier, more resilient, and resource-efficient built environments for future generations.

Open Challenges and Possible Solutions

Computational Complexity

Generative models, while offering remarkable capabilities in generating data similar to a given training set, often exhibit significant computational complexity. This complexity arises due to several factors inherent in their architecture and training process. Firstly, the intricate network architectures of generative models, such as Diffusion Model [52], Generative Adversarial Networks (GANs) [7], and Autoregressive Models [9], contribute to computational demands. These models typically comprise numerous layers and parameters, necessitating extensive computations during both training and inference phases. Additionally, the optimization algorithms used for training, such as stochastic gradient descent [113] or variants like Adam [114], require iterative calculations over large datasets, further increasing computational load. Furthermore, the scale and complexity of the training data also play a crucial role; larger datasets demand more computational resources for processing.

In order to tackle the problem of computational complexity in generative models, several strategies can be employed. One approach involves optimizing the model architecture itself by designing more efficient and streamlined network structures, reducing the number of parameters [115], or utilizing model compression techniques like knowledge distilation [116], pruning [117, 118] or quantization [119]. This optimization allows for a reduction in computational complexity without compromising performance. Another method is parallelization and distributed computing, where generative models take advantage of parallel computing architectures such as GPUs or distributed systems to perform computations simultaneously [120]. By distributing the workload, the training and inference processes can be accelerated, leading to more efficient generative modeling.

Additionally, approximate inference methods like Bayesian inference [121], or Monte Carlo sampling [122] can be used to approximate complex probability distributions, providing fast and tractable approximations when exact inference is computationally infeasible. Pre-training models on large datasets or employing transfer learning techniques [123] can help overcome computational complexity by leveraging learned representations and reducing the computational burden during training or inference. Finally, optimization algorithms and heuristics like stochastic gradient descent or early stopping can optimize the model parameters and improve convergence, reducing computational overhead [124]. By employing these strategies, researchers and practitioners can effectively address computational complexity in generative models, making them more efficient, practical, and accessible for a wide range of applications.

Handling Large-Scale Floorplans

With the growing demand for large and complex architectural projects, floorplan generation methods must adapt to handle the scaling requirements. Handling large-scale floorplans presents a challenge in floorplan generation due to increased computational complexity and the availablity of training data [19]. As floorplans grow in size and complexity, the number of elements, relationships, and details to be considered escalates significantly, demanding more memory and processing power. This can lead to longer training times, higher resource requirements, and slower sampling, hindering the efficiency of generative models.

To address the challenge of handling large-scale floorplans, potential solutions include model optimization, which entails designing efficient neural network architectures, employing model distillation, or implementing pruning techniques to reduce model complexity. Model optimization techniques, as highlighted in [125], focus on refining neural network architectures to better suit the requirements of handling large-scale floorplans. By streamlining the architecture and reducing unnecessary computational overhead, these methods aim to improve efficiency without compromising performance. Model distillation, as discussed in [126], involves training a smaller, distilled model to mimic the behavior of a larger, more complex model. This approach can help reduce memory and processing requirements while retaining the generative capabilities of the original model. Additionally, pruning techniques, as outlined in [117, 118], involve removing redundant connections or parameters from the neural network to reduce its size and complexity. By eliminating unnecessary components, pruning techniques can lead to more efficient inference and sampling processes.

Moreover, adopting a progressive generation approach, as suggested in [110], can be beneficial for handling large-scale floorplans. This strategy involves generating floorplans incrementally, starting with a coarse layout and iteratively refining details. By breaking down the generation process into smaller, more manageable steps, progressive generation promotes faster convergence and enhances overall sampling efficiency. This iterative refinement allows the model to focus on capturing finer details as it progresses, leading to more realistic and coherent floorplan designs. Additionally, by generating floorplans in stages, this approach can help mitigate the computational complexity associated with processing large-scale datasets, making it well-suited for addressing the challenges of handling complex architectural projects.

Ensuring Diversity in Generated Floorplans

The challenge in diverse floorplan generation lies in accomplishing the task of producing a broad spectrum of visually unique and original floorplans from a given distribution. Here are some of the key technical reasons:

A. Mode collapse

Generative models like GANs [7] face challenges such as vanishing gradients and mode collapse during training. GAN model collapse is characterized by the generator's failure to produce diverse and meaningful outputs. where only a few sample modes are generated. This is evident through a near-zero gradient norm ($\|\nabla_{\theta_g} L_G\| \approx 0$) in the generator's loss (L_G) with respect to its parameters (θ_g), leading to slow learning. As a consequence, parameter updates are restricted, hindering exploration of the entire data space and leading to single-mode approximations ($p_g(x) \approx \delta(x - M)$), severely limiting sample diversity. Basically, GAN mode collapse can occur due to factors such as an imbalance between the generator and discriminator ($D(G(z)) \approx 0.5$), mode collapse ($G(z) \approx G(z')$), lack of exploration in the latent space (limited z variations), gradient vanishing, and training instability (oscillating loss functions).

Architectural modifications are pivotal in mitigating the issue of mode collapse encountered in Generative Adversarial Networks (GANs). By enhancing the depth of the generator network, as suggested in [127], GANs can produce more diverse and intricate outputs, thereby reducing the tendency to overfit on a limited set of modes. Multi-scale architectures, as explored in [128], prove effective in capturing both global and local features, thus promoting diversity and expanding the range of generated samples. Furthermore,

Conditional GANs (CGANs), as discussed in [76], integrate additional input features to guide the generator towards specific modes, thereby mitigating mode collapse by providing more explicit control over the generated outputs. Additionally, regularization techniques such as weight decay, dropout, and batch normalization, as outlined in [129], are instrumental in preventing overfitting and encouraging exploration of different modes, thus fostering diversity in the generated samples. Moreover, training strategies such as alternating learning rates or updating frequencies, as suggested in various sources, contribute to the stability and convergence of GAN training, further alleviating the limitations posed by mode collapse. By implementing these architectural modifications and training strategies, GANs can effectively overcome mode collapse limitations, generate a wider range of diverse and realistic samples, and consequently address the issue of diversity in the generated outputs.

B. Optimization techniques

Optimization techniques play a significant role in influencing the diversity of generated outputs in various generative methods. The choice of optimization algorithms and strategies can impact how a generative model explores the data space and generates diverse samples. When optimization is too aggressive or constrained, it may prevent the model from adequately exploring different modes of data distribution, leading to reduced diversity in generated samples. Conversely, well-tailored optimization methods can promote a smoother convergence, allowing the model to capture a broader range of patterns and modes, thereby enhancing diversity in the generated outputs. Techniques such as gradient clipping [130], weight regularization and learning rate scheduling can affect the convergence behavior of the model.

To address the challenges associated with optimization techniques and promote diversity in generated outputs, several solutions can be implemented. One approach is to explore advanced optimization algorithms such as evolutionary algorithms [131] or Bayesian optimization [132], which can provide a more robust exploration of the data space and help overcome local optima. Additionally, incorporating regularization techniques [133] like dropout or batch normalization can introduce controlled noise during training, encouraging the model to explore different modes and increasing output diversity. Another strategy involves adjusting the learning rate dynamically during training, using techniques like learning rate annealing [134] to strike a balance between exploration and convergence. Furthermore, promoting diversity can be achieved by incorporating diversity-inducing objectives or introducing specific constraints on the model's parameters to encourage exploration. By carefully selecting and combining these solutions, generative models can produce diverse and high-quality outputs, addressing the challenges associated with optimization techniques.

Handling Non-Regular Shapes

Generating irregular floorplans poses a challenge due to the lack of existing data with non-regular layouts. The currently available floorplan datasets predominantly consist of regular and common patterns, making it difficult for generative models to learn and generate irregular designs [15]. This limitation arises because the models tend to rely on the patterns and biases present in the training data. As a result, creating floorplans with irregular shapes requires innovative approaches that go beyond the existing dataset limitations.

To address the challenge of generating irregular shapes in floorplan layouts and enhance a generative model's capacity for diversity and creativity, several strategies can be effectively combined. One key strategy is to enrich the training dataset with a diverse range of irregular floorplan layouts, exposing the model to a wider variety of configurations. By including floorplans with non-standard shapes, such as irregular polygons or curved walls, the model can learn to generate more diverse and unconventional layouts. Another approach is to employ data augmentation techniques [135], such as random transformations and deformations, during the training process. These techniques introduce artificial variations that simulate irregularities in real-world floorplan layouts. By applying random rotations, translations, or distortions to the input data, the model can learn to generate irregular shapes that go beyond the limitations of regular grid-based layouts. Conditional generative models [16] have also been utilized for irregular floorplan generation. These models take into account specific constraints or input features that guide the generation process. By incorporating constraints related to room sizes, connectivity, or specific architectural requirements, the generative model can produce irregular shapes that align with desired criteria. This allows for more precise control over the generated layouts while still promoting diversity and creativity. Moreover, hybrid approaches that combine generative models with procedural methods specialized in handling irregular shapes can be employed. Procedural methods, such as shape grammars or procedural modeling techniques [136], excel at generating irregular shapes and intricate details. By integrating these methods with generative models, the strengths of both techniques can be leveraged, resulting in more accurate and creative floorplan layouts with irregular shapes.

Compatibility

Architectural design encompasses subjective decisions that are heavily influenced by individual user preferences. To create floorplans that truly connect with people, it becomes crucial to incorporate user feedback and preferences throughout the generation process, despite the added complexity. Among the various factors affecting this interaction, compatibility emerges as a pivotal element [19]. Addressing compatibility issues becomes essential to strike a harmonious balance between user preferences and architectural practicality, ensuring that the resulting spaces not only reflect personal tastes but also function seamlessly within the overall environment. By tackling compatibility challenges, floorplan designs can achieve a cohesive integration of user preferences and architectural considerations, leading to spaces that are both aesthetically pleasing and functionally efficient. Let:

I : Input modality (bubble diagram, text, image or other.)

G : Generated floorplan.

C : Compatibility metric between input and generated floorplan.

The compatibility metric (C) can be conceptualized as a function of several factors:

1. **Semantic Alignment (SA)**: How well the semantics of the input match the design elements in the floorplan.
2. **Spatial Alignment (SPA)**: How closely the spatial arrangement of design elements in the floorplan matches the intended layout from the input.
3. **Constraint Adherence (CA)**: The extent to which the generated floorplan adheres to various design constraints, including structural, functional, and regulatory requirements.
4. **Expressiveness (EX)**: The ability of the input modality to capture the complexity and details required for accurate floorplan generation.
5. **Information Completeness (IC)**: The degree to which the input provides all necessary information to generate a complete and accurate floorplan.

The compatibility metric (C) between the input modality (I) and the generated floorplan (G) is given by:

$$\begin{aligned} C(I,G) = {} & w_{SA} \cdot \mathrm{SA}(I,G) + w_{SPA} \cdot \mathrm{SPA}(I,G) \\ & + w_{CA} \cdot \mathrm{CA}(G) + w_{EX} \cdot \mathrm{EX}(I) + w_{IC} \cdot \mathrm{IC}(I) \end{aligned} \tag{12}$$

Here, w_{SA}, w_{SPA}, w_{CA}, w_{EX}, and w_{IC} are weighting coefficients that determine the relative importance of each factor. These weights are assigned based on the specific requirements and priorities of the design process.

To improve the compatibility between input modalities and generated floorplans in floorplan generation, enhancing data representation is crucial. This entails refining the way input modalities such as textual descriptions are presented, either by offering more detailed descriptions or by incorporating multiple modalities for a more comprehensive input [137]. By enhancing data representation, the model gains a clearer understanding of the relationships between different modalities, thus facilitating more accurate floorplan generation. Additionally, employing multi-modal learning techniques enables the model to leverage information from various modalities simultaneously, enhancing its ability to capture intricate relationships and generate floorplans that better align with the input data [137].

Integrating attention mechanisms into the generation process enables the model to focus on pertinent aspects of the input modalities, ensuring that the resulting floorplans accurately represent the contained information. This attention-based strategy enhances the model's ability to discern crucial details and incorporate them into the generated output [138]. Implementing a feedback loop, where the generated floorplans are continuously evaluated against input modalities and used to refine the model iteratively, further improves compatibility. Through this iterative refinement process, the model learns to better align the generated floorplans with the input modalities, ultimately enhancing the quality and fidelity of the output over time. Combining vectorized generation with bubble diagram modality, as advocated by [19], emerges as a highly effective approach for enhancing interactivity accessibility. This integrated method offers a dependable solution for creating interactive floorplans that are both accessible and engaging.

Balancing Customization and Quality in Floorplan Generations

Customization serves as a crucial aspect in floorplan generation, enabling layouts to be finely tuned to meet specific requirements and preferences. Despite its significance, numerous generative models employed in this domain are trained on and produce raster floorplans, presenting considerable obstacles for direct customization due to inherent limitations in raster data representation [16]. Raster-based floorplans lack the structural flexibility necessary for seamless adaptation to individualized needs, hindering efficient customization efforts. Consequently, while generative models excel in generating initial layouts, the transition to tailored designs confronts hurdles stemming from the constrained nature of raster data.

Addressing the limitations posed by raster-based floorplan generation, the field often turns to post-processing techniques [16], yet these methods may introduce drawbacks affecting the quality of generated floorplans. To overcome

these challenges, the exploration of end-to-end vectorization methods [20] gains prominence, as they operate with vector data, offering enhanced flexibility and precision in customization. However, the development of such methods entails complexities in data transformations. Balancing customization with high-quality results remains a complex challenge in floorplan generation.

And the integration of quality control mechanisms within generative models, serving as safeguards to ensure that customization aligns with user preferences and design constraints without compromising the overall quality, functionality, and coherence of the generated floorplans. Prioritizing research in vectorization and quality control holds the key to empowering users to customize floorplans while ensuring optimal outcomes.

Ensuring Realism for Generated Floorplans

Ensuring realism in generated floorplans is a complex task involving the creation of architectural layouts that closely resemble real-world designs. Realism encompasses various aspects such as architectural accuracy, functional considerations, aesthetic appeal, room proportions, spatial flow, and compliance with regulations. Unrealistic floorplans may exhibit issues like improper room size and shape, non-connected rooms, jagged boundaries, and broken lines. To address these challenges, various computational methods and techniques can be employed.

One approach to achieving realism in generated floorplans is through leveraging architectural modifications using graph neural networks [16]. By representing floorplans as graphs and utilizing graph neural networks, it becomes possible to capture spatial relationships and generate layouts that adhere to architectural principles. This method enables the model to understand the connections between different rooms and elements within the floorplan, resulting in more coherent and accurate designs. User feedback and iterative design play a crucial role in enhancing realism. By involving users in the generation process and incorporating their preferences and feedback, the floorplans can be iteratively refined to better align with real-world design standards and meet users' expectations. This iterative approach ensures that the generated floorplans are tailored to individual needs and preferences, resulting in more realistic and user-centric designs.

Another technique is procedural generation [136], which offers a way to create diverse and realistic floorplans. Procedural generation algorithms can automatically generate a large number of floorplans with varying room arrangements, spatial flows, and aesthetic appeal while still adhering to functional and regulatory constraints [78]. By incorporating attention mechanisms [102], such as those used in transformer models, the generation process can focus on relevant architectural details and capture dependencies and long-range interactions within the floorplans, ensuring coherence and realism. Sequential generation models, such as diffusion models [52] or autoregressive models [9], allow for step-by-step generation of floorplans while considering dependencies between different parts of the layout. These models capture complex spatial relationships and can generate realistic floorplans by sequentially adding rooms or modifying existing ones. Increasing the amount of data available for training is also crucial for improving realism. By expanding the dataset of real floorplans used during the training phase, models can learn from a wider range of architectural designs and patterns, resulting in more realistic and diverse generated floorplans that align with real-world architectural standards and aesthetics. By leveraging computational methods like architectural modifications, user feedback, procedural generation, attention mechanisms, sequential generation models, and increasing the training data, it becomes possible to generate floorplans that closely resemble real-world designs while adhering to architectural principles, functional considerations, and aesthetic appeal.

Future Research Directions

The rapid advancement of computational methods across various real-time industries has notably influenced the architectural sector as well. With technology continually evolving, the integration of computational techniques into architectural practices has become increasingly prevalent. In response to this trend, numerous avenues for future research in floorplan generation using computational-based methods have emerged. By examining the intersection of computational techniques and architectural innovation, this study seeks to identify key areas ripe for exploration and advancement, paving the way for the development of more efficient, sustainable, and customizable floorplan generation processes.

- *Exploring Modalities for Diverse User Engagement:* Future research in floorplan generation could explore innovative modalities to cater to diverse user needs, including non-domain experts and individuals with disabilities. By embracing different modalities, such as natural language processing, gesture recognition, or multimodal interfaces, computational tools can empower a broader range of users to participate in the design process and customize floorplans according to their preferences and requirements. Moreover, incorporating accessibility features and design guidelines tailored to individuals with disabilities can ensure that generated floorplans are inclusive and barrier-free.

This research could involve developing algorithms that dynamically adjust floorplan layouts based on real-time input from users, resulting in more personalized and satisfactory designs. By adopting a multifaceted approach that considers various modalities and user perspectives, future research has the potential to democratize architectural design and create more inclusive and user-centric built environments.

- *Energy Efficiency Modeling:*

  Future research in energy-efficient floorplan generation using AI holds promise for revolutionizing sustainable architecture. By harnessing the capabilities of AI algorithms, researchers can explore novel approaches to optimize building layouts and spatial configurations to minimize energy consumption and maximize efficiency. This research could involve the development of AI models that integrate data on building orientation, thermal performance, and occupant behavior to inform the design process and generate floorplans that prioritize energy efficiency. Additionally, AI-driven simulations and predictive analytics can be employed to evaluate the performance of generated floorplans under different environmental conditions and usage scenarios, allowing architects to make informed decisions that lead to more sustainable building designs. Overall, advancing the intersection of AI and energy-efficient floorplan generation offers significant potential for reducing carbon footprints and creating buildings that are environmentally friendly and cost-effective in the long term.
- *Balancing efficiency and diversity:*

  Future research in floorplan generation could focus on optimizing the balance between sampling time and the diversity of generated floorplans. This entails developing efficient algorithms that can quickly explore a wide range of design possibilities while ensuring that the resulting floorplans maintain diversity and creativity. One approach could involve leveraging techniques from evolutionary algorithms or reinforcement learning to dynamically adjust the exploration-exploitation trade-off during the generation process. Additionally, researchers could explore the use of surrogate models or parallel computing to accelerate the sampling process without sacrificing the quality or diversity of the generated designs. By addressing this challenge, future research has the potential to streamline the floorplan generation process and enable architects to efficiently explore a diverse range of design options to meet various project requirements and constraints.
- *Ethical and Societal Implications:*

  Future research should delve into the ethical and societal implications arising from AI-generated floorplans, delving into critical issues such as privacy, equity, and cultural sensitivity. By scrutinizing these aspects, researchers can pave the way for the responsible implementation of AI technologies in architectural practice. This involves developing comprehensive guidelines to navigate complex ethical considerations, ensuring that AI-generated floorplans prioritize privacy protection, promote equity in access to design resources, and respect diverse cultural perspectives. Moreover, this research seeks to foster a deeper understanding of the broader societal impacts of AI in architecture, facilitating informed decision-making and promoting the development of ethically sound practices that benefit all stakeholders.
- *3D floorplan Generation:*

  3D floorplan generation is an exciting area of research aimed at expanding the capabilities of floorplan generation by incorporating the third dimension. By moving beyond traditional 2D representations, 3D floorplan generation aims to provide a more immersive and realistic depiction of architectural spaces. This involves considering factors such as room height, furniture placement, and spatial arrangement in the vertical dimension. By incorporating the third dimension, AI models can generate floorplans that capture the true volume and depth of a space, allowing architects, designers, and users to better visualize and understand the layout. This advancement opens up possibilities for more accurate and detailed representations of architectural designs, enabling better decision-making, enhanced spatial planning, and improved communication between stakeholders involved in the design and construction process. The exploration of 3D floorplan generation holds great potential in revolutionizing the way architectural spaces are conceptualized, designed, and experienced.

Conclusion

House floorplan generation is a complex endeavor that balances functionality and aesthetics in interior layouts. Automation in this process accelerates design, reduces errors, and offers creative exploration. In this article, we address the computational methods which aim to synthesize floorplans. We provide a comprehensive and novel classification of computational-based floorplan generation into subcategories of Procedural and machine-learning methods. Representation and interactivity methods were also discussed in the review. For the recent developments in the area, we identify the types, key features, representations, and algorithms in each category. Furthermore, we propose general sampling and training time complexity for each deep learning model and present comparative analysis with cross-over. Detailed

information on the characteristics of state-of-the-art solutions for each category, their advantages, and drawbacks are provided.

While there are existing solutions for various dimensions of floorplan generation, the field still faces open research issues, challenges, and areas in need of improvement. In this article, we not only present state-of-the-art methods for computational-based floorplan generation, including their key properties, classification, and algorithms but also delve into open questions and research directions within this domain. In the Challenges and Possible Directions section, we delve into the computational aspects and highlight the significance of incorporating diversity and realism into the generated floorplans. We also discuss the need for effectively managing irregularities and taking compatibility issues into account.

This comprehensive overview provides architects with a valuable resource to enhance their design process, improve efficiency, and create innovative spaces using computational floorplan generation techniques. It also empowers individuals to customize their own floor plans, fostering a sense of ownership. On a societal level, adopting these techniques leads to enhanced resource utilization, improved energy efficiency, optimized space utilization, and better living and working environments. The review identifies opportunities for future development, further enhancing the societal benefits of computational floorplan generation. Ultimately, policymakers, researchers, and professionals can utilize this review to make informed decisions and contribute to sustainable and functional spaces for diverse communities.

Author Contributions All authors have contributed equally in conceptual work, validation, and write-up.

Funding Not applicable.

Data Availability Not applicable because the paper is a survey paper.

Declarations

Conflict of interest There are no financial or non-financial interests that are directly or indirectly related to this work.

Human or Animals Rights Not applicable because no animal or human is involved in this paper.

Informed Consent We have not used any information that needs consent from any person or organization.

References

1. Heckmann O, Schneider F. Floor plan manual: Housing. 1997.
2. Brooks RA, Lozano-Perez T. A subdivision algorithm in configuration space for findpath with rotation. IEEE Trans Syst Man Cybern. 1985;2:224–33.
3. Rinde L, Dahl A. Procedural generation of indoor environments. Charmers University of Technology 2008.
4. Peng C-H, Yang Y-L, Wonka P. Computing layouts with deformable templates. ACM Trans Graph (TOG). 2014;33(4):1–11.
5. Martin J. Procedural house generation: A method for dynamically generating floor plans. In: Proceedings of the Symposium on Interactive 3D Graphics and Games, 2006;2.
6. Merrell P, Schkufza E. Koltun V. Computer-generated residential building layouts. In: ACM SIGGRAPH Asia 2010 Papers, 2010;1–12.
7. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y. Generative adversarial nets. Adv. Neural Inf. Process. Syst. 2014;27.
8. Chai S, Zhuang L, Yan F. Layoutdm: Transformer-based diffusion model for layout generation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023;18349–18358.
9. Para W, Guerrero P, Kelly T, Guibas LJ, Wonka P. Generative layout modeling using constraint graphs. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021;6690–6700.
10. Pavie N, Gilet G, Dischler J-M, Ghazanfarpour D. Procedural texture synthesis by locally controlled spot noise 2016.
11. Salakhutdinov R. Learning deep generative models. Ann Rev Stat Appl. 2015;2:361–85.
12. Wang S, Zeng W, Chen X, Ye Y, Qiao Y, Fu C-W. Actfloor-gan: Activity-guided adversarial networks for human-centric floorplan design. IEEE Trans Visual Comput Graph. 2021;1:1.
13. Chaillou S. Archigan: a generative stack for apartment building design. NVIDIA Corporation; 2019.
14. Tang H, Zhang Z, Shi H, Li B, Shao L, Sebe N, Timofte R, Van Gool L. Graph transformer gans for graph-constrained house generation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023;2173–2182
15. Nauata N, Chang K-H, Cheng C-Y, Mori G, Furukawa Y. House-gan: Relational generative adversarial networks for graph-constrained house layout generation. In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16, 2020;162–177. Springer.
16. Nauata N, Hosseini S, Chang K, Chu H, Cheng C, Furukawa Y. House-gan++: Generative adversarial layout refinement network towards intelligent computational agent for professional architects. In: CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021;13627–13636.
17. Liu Y, Luo Y, Deng Q, Zhou X. Exploration of campus layout based on generative adversarial network: Discussing the significance of small amount sample learning for architecture. In: Proceedings of the 2020 DigitalFUTURES: The 2nd International Conference on Computational Design and Robotic Fabrication (CDRF 2020), 2021;169–178. Springer.
18. Chailou S. Space layouts & gans | gan-enabled floor plan generation. Towards Data Science 2020.
19. Shabani MA, Hosseini S, Furukawa Y. Housediffusion: Vector floorplan generation via a diffusion model with discrete and continuous denoising. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023;5466–5475.
20. Liu J, Xue Y, Duarte J, Shekhawat K, Zhou Z, Huang X. End-to-end graph-constrained vectorized floorplan generation with panoptic refinement. In: European Conference on Computer Vision, 2022;547–562. Springer.
21. Liu C, Wu J, Furukawa Y. Floornet: A unified framework for floorplan reconstruction from 3d scans. In: Proceedings of the European Conference on Computer Vision (ECCV), 2018;201–217.

22. Hu R, Huang Z, Tang Y, Van Kaick O, Zhang H, Huang H. Graph2plan: Learning floorplan generation from layout graphs. ACM Trans Graph (TOG). 2020;39(4):118.
23. Chen Q, Wu Q, Tang R, Wang Y, Wang S, Tan M. Intelligent home 3d: Automatic 3d-house design from linguistic descriptions only. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020;12625–12634.
24. Živković M, Žujović M, Milošević J. Architectural 3d-printed structures created using artificial intelligence: A review of techniques and applications. Appl Sci. 2023;13(19):10671.
25. Zhang Z, Fort JM, Mateu LG. Exploringthe potential of artificial intelligence as a tool for architectural design: A perception study using gaudí'sworks. Buildings. 2023;13(7):1863.
26. Caldas L. Generation of energy-efficient architecture solutions applying gene_arch: An evolution-based generative design system. Adv Eng Inform. 2008;22(1):59–70.
27. Wang D, Liu K, Johnson P, Sun L, Du B, Fu Y. Deep human-guided conditional variational generative modeling for automated urban planning. In: 2021 IEEE International Conference on Data Mining (ICDM), 2021;679–688. IEEE.
28. Shi Y, Shang M, Qi Z. Intelligent layout generation based on deep generative models: a comprehensive survey. Inf Fus, 2023;101940.
29. Abd El-Maksoud NM, Ahmed EB. Artificial intelligence applications in green architecture. Fayoum Univ J Eng. 2024;7(2):317–37.
30. Rane N, Choudhary S, Rane J. Leading-edge technologies for architectural design: a comprehensive review. Available at SSRN 4637891 2023.
31. ZHENG H, Keyao A, Jingxuan W, Yue R. Apartment floor plans generation via generative adversarial networks. In: 25th International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA 2020): RE: Anthropocene, Design in the Age of Humans, 2020;601–610. The Association for Computer-Aided Architectural Design Research in Asia ...
32. Tzelepi M, Nousi P, Passalis N, Tefas A. Representation learning and retrieval. Deep Learning for Robot Perception and Cognition 2022.
33. Isola P, Zhu J-Y, Zhou T, Efros AA. Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017;1125–1134
34. Zhu J-Y, Park T, Isola P, Efros AA. Unpaired image-to-image translation using cycle-consistent adversarial networks. In: Proceedings of the IEEE International Conference on Computer Vision, 2017;2223–2232
35. Ramesh A, Pavlov M, Goh G, Gray S, Voss C, Radford A, Chen M, Sutskever I. Zero-shot text-to-image generation. In: International Conference on Machine Learning, 2021;8821–8831. PMLR.
36. Ding M, Zheng W, Hong W, Tang J. Cogview2: Faster and better text-to-image generation via hierarchical transformers. Adv Neural Inf Process Syst. 2022;35:16890–902.
37. Bowman SR, Vilnis L, Vinyals O, Dai AM, Jozefowicz R, Bengio S. Generating sentences from a continuous space. 2015. arXiv preprint arXiv:1511.06349.
38. Liu PJ, Saleh M, Pot E, Goodrich B, Sepassi R, Kaiser L, Shazeer N. Generating wikipedia by summarizing long sequences. 2018. arXiv preprint arXiv:1801.10198.
39. Schick T, Schütze H. Few-shot text generation with natural language instructions. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, 2021;390–402.
40. Sharma D, Dhiman C, Kumar D. Automated image caption generation framework using adaptive attention and bi-lstm. In: 2022 IEEE Delhi Section Conference (DELCON), 2022;1–5. IEEE.
41. Para W, Guerrero P, Kelly T, Guibas LJ, Wonka P. Generative layout modeling using constraint graphs. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021;6690–6700
42. Chen A. Generation of layouts for living spaces using conditional generative adversarial networks: Designing floor plans that respect both a boundary and high-level requirements 2022.
43. Emmons P. The cosmogony of bubble diagrams. In: 86th ACSA Annual Meeting and Technology Conference, Constructing Identity, 1998;420–425.
44. Lim H. Automatic generation of ai-powered architectural floor plans using grid data. Int J Appl Eng Res. 2023;18(2):97–102.
45. Sonbol R, Rebdawi G, Ghneim N. The use of nlp-based text representation techniques to support requirement engineering tasks: A systematic mapping review. IEEE Access. 2022;10:62811–30.
46. Liu H, Cui L, Liu J, Zhang Y. Natural language inference in context-investigating contextual reasoning over long texts. In: Proceedings of the AAAI Conference on Artificial Intelligence, 2021;35:13388–13396.
47. Tutenel T, Bidarra R, Smelik RM, De Kraker KJ. Rule-based layout solving and its application to procedural interior generation. In: CASA Workshop on 3D Advanced Media in Gaming and Simulation 2009.
48. Froumentin M, Labrosse F, Willis P. A vector-based representation for image warping. In: Computer Graphics Forum, 2000;19:419–425. Wiley Online Library.
49. Athiwaratkun B, Kang K. Feature representation in convolutional neural networks. 2015. arXiv preprint arXiv:1507.02313.
50. Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G. The graph neural network model. IEEE Trans Neural Netw. 2008;20(1):61–80.
51. Schiller E. Creating novel architectural layouts with generative adversarial networks. PhD thesis, Harvard University 2018.
52. Ho J, Jain A, Abbeel P. Denoising diffusion probabilistic models. Adv Neural Inf Process Syst. 2020;33:6840–51.
53. Borgelt C. An implementation of the fp-growth algorithm. In: Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations, 2005;1–5.
54. Smelik RM, Tutenel T, Bidarra R, Benes B. A survey on procedural modelling for virtual worlds. Comput Graph Forum. 2014;33:31–50.
55. Braben D, Bell I (1984) Elite. Frontier Developments. http://frontier.co.uk/games/elite
56. Bulbul A. Procedural generation of semantically plausible small-scale towns. Graph Models. 2023;126:101–70.
57. Parish YI, Müller P. Procedural modeling of cities. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, 2001;301–308.
58. Bavoil L, Myers K. Order independent transparency with dual depth peeling. NVIDIA OpenGL SDK. 2008;1:12.
59. Borland D, Clarke JP, Fielding JR, Taylor RM II. Volumetric depth peeling for medical image display. Visual Data Anal. 2006;6060:35–45.
60. Liu F, Huang M-C, Liu X-H, Wu E-H. Efficient depth peeling via bucket sort. In: Proceedings of the Conference on High Performance Graphics 2009, 2009;51–57.
61. Serdar T, Sechen C. Automatic datapath tile placement and routing. In: Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001, 2001;552–559. IEEE.
62. Kaplan C. Introductory Tiling Theory for Computer Graphics. Morgan & Claypool Publishers; 2009.

63. Aldino AA, Pratiwi ED, Sintaro S, Putra AD, et al. Comparison of market basket analysis to determine consumer purchasing patterns using fp-growth and apriori algorithm. In: 2021 International Conference on Computer Science, Information Technology, and Electrical Engineering (ICOMITEE), 2021;29–34. IEEE.
64. Xu F, Lu H. The application of fp-growth algorithm based on distributed intelligence in wisdom medical treatment. Int J Pattern Recognit Artif Intell. 2017;31(04):1759005.
65. Hahn E, Bose P, Whitehead A. Persistent realtime building interior generation. In: Proceedings of the 2006 ACM SIGGRAPH Symposium on Videogames, 2006;179–186
66. Marson F, Musse SR. Automatic real-time generation of floor plans based on squarified treemaps algorithm. Int J Comput Games Technol. 2010;2010:1–10.
67. Tutenel T, Bidarra R, Smelik RM, De Kraker KJ. Rule-based layout solving and its application to procedural interior generation. In: CASA Workshop on 3D Advanced Media in Gaming and Simulation 2009.
68. Lopes R, Tutenel T, Smelik RM, De Kraker KJ, Bidarra R. A constrained growth method for procedural floor plan generation. In: Proc 11th Int Conf Intell Games Simul, 2010;13–20. Citeseer.
69. Zhao R, Ji Q. An adversarial hierarchical hidden markov model for human pose modeling and generation. In: Proceedings of the AAAI Conference on Artificial Intelligence, 2018;32.
70. Cao Y, Sun L, Han C, Guo J. Improved side information generation algorithm based on naive bayesian theory for distributed video coding. IET Image Proc. 2018;12(3):354–60.
71. Fernando B, Fromont E, Muselet D, Sebban M. Supervised learning of gaussian mixture models for visual vocabulary generation. Pattern Recogn. 2012;45(2):897–907.
72. Sutton C, McCallum A, et al. An introduction to conditional random fields. Found Trends® Mach Learn. 2012;4(4):267–373.
73. LaValley MP. Logistic regression. Circulation. 2008;117(18):2395–9.
74. Park N, Mohammadi M, Gorde K, Jajodia S, Park H, Kim Y. Data synthesis based on generative adversarial networks. 2018. arXiv preprint arXiv:1806.03384.
75. Ziegler JD, Subramaniam S, Azzarito M, Doyle O, Krusche P, Coroller T. Multi-modal conditional gan: Data synthesis in the medical domain. In: NeurIPS 2022 Workshop on Synthetic Data for Empowering ML Research 2022.
76. Mirza M, Osindero S. Conditional generative adversarial nets. 2014. arXiv preprint arXiv:1411.1784.
77. Radford A, Metz L, Chintala S. Unsupervised representation learning with deep convolutional generative adversarial networks. 2015. arXiv preprint arXiv:1511.06434.
78. Karras T, Aila T, Laine S, Lehtinen J. Progressive growing of gans for improved quality, stability, and variation. 2017. arXiv preprint arXiv:1710.10196.
79. Arjovsky M, Chintala S, Bottou L. Wasserstein generative adversarial networks. In: International Conference on Machine Learning, 2017;214–223. PMLR.
80. Chen X, Duan Y, Houthooft R, Schulman J, Sutskever I, Abbeel P. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. Adv Neural Inf Process Syst. 2016;29.
81. Karras T, Laine S, Aila T. A style-based generator architecture for generative adversarial networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019;4401–4410.
82. Li C, Wang Z, Qi H. Fast-converging conditional generative adversarial networks for image synthesis. In: 2018 25th IEEE International Conference on Image Processing (ICIP), 2018;2132–2136. IEEE.
83. Zhang H, Xu T, Li H, Zhang S, Wang X, Huang X, Metaxas DN. Stackgan++: Realistic image synthesis with stacked generative adversarial networks. IEEE Trans Pattern Anal Mach Intell. 2018;41(8):1947–62.
84. Li X, Luo J, Younes R. Activitygan: Generative adversarial networks for data augmentation in sensor-based human activity recognition. In: Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers, 2020;249–254
85. Ledig C, Theis L, Huszár F, Caballero J, Cunningham A, Acosta A, Aitken A, Tejani A, Totz J, Wang Z, et al. Photo-realistic single image super-resolution using a generative adversarial network. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017;4681–4690.
86. Chen Z, Chen X. Tachiegan: Generative adversarial networks for tachie style transfer. In: 2022 IEEE International Conference on Multimedia and Expo Workshops (ICMEW), 2022;1–6. IEEE.
87. Pektas M, Gecer B, Ugur A. Efficient hair style transfer with generative adversarial networks. 2022. Xiv preprint arXiv:2210.12524.
88. Zhang Y, Yin Y, Zimmermann R, Wang G, Varadarajan J, Ng S-K. An enhanced gan model for automatic satellite-to-map image conversion. IEEE Access. 2020;8:176704–16.
89. Shum KC, Pang H-W, Hua B-S, Nguyen DT, Yeung S-K. Conditional 360-degree image synthesis for immersive indoor scene decoration. 2023. Xiv preprint arXiv:2307.09621.
90. Wu W, Fu X-M, Tang R, Wang Y, Qi Y-H, Liu L. Data-driven interior plan generation for residential buildings. ACM Trans Graph (TOG). 2019;38(6):1–12.
91. Hsu Y-C, Fontaine M, Earle S, Edwards M, Togelius J, Nikolaidis S. Generating diverse indoor furniture arrangements. In: ACM SIGGRAPH 2022 Posters, 2022;1-2
92. Zhang Y, Yin Y, Zimmermann R, Wang G, Varadarajan J, Ng S-K. An enhanced gan model for automatic satellite-to-map image conversion. IEEE Access. 2020;8:176704–16.
93. Chang K-H, Cheng C-Y, Luo J, Murata S, Nourbakhsh M, Tsuji Y. Building-gan: Graph-conditioned architectural volumetric design generation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021;1956–11965.
94. LaValle S. Rapidly-exploring random trees: A new tool for path planning. Research Report 9811 1998.
95. Zhang F, Nauata N, Furukawa Y. Conv-mpn: Convolutional message passing neural network for structured outdoor architecture reconstruction. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020;2798–2807.
96. Singh J, Gould S, Zheng L. High-fidelity guided image synthesis with latent diffusion models. In: 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023;5997–6006. IEEE.
97. Zhang Y, Huang N, Tang F, Huang H, Ma C, Dong W, Xu C. Inversion-based style transfer with diffusion models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023;10146–10156.
98. Li Y, Wang H, Jin Q, Hu J, Chemerys P, Fu Y, Wang Y, Tulyakov S, Ren J. Snapfusion: Text-to-image diffusion model on mobile devices within two seconds. 2023. arXiv preprint arXiv:2306.00980.
99. Alexanderson S, Nagy R, Beskow J, Henter GE. Listen, denoise, action! audio-driven motion synthesis with diffusion models. ACM Trans Graph (TOG). 2023;42(4):1–20.
100. Nichol AQ, Dhariwal P. Improved denoising diffusion probabilistic models. In: International Conference on Machine Learning, 2021;8162–8171. PMLR.

101. He L, Lu Y, Corring J, Florencio D, Zhang C. Diffusion-based document layout generation. 2023. arXiv preprint arXiv:2303.10787.
102. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. Adv Neural Inf Process Syst. 2017;30.
103. Yu J, Xu Y, Koh JY, Luong T, Baid G, Wang Z, Vasudevan V, Ku A, Yang Y, Ayan BK, et al. Scaling autoregressive models for content-rich text-to-image generation. 2022;2(3):5. arXiv preprint arXiv:2206.10789.
104. Yu J, Xu Y, Koh JY, Luong T, Baid G, Wang Z, Vasudevan V, Ku A, Yang Y, Ayan BK, et al. Scaling autoregressive models for content-rich text-to-image generation. 2022;2(3):5. arXiv preprint arXiv:2206.10789.
105. Oord A, Kalchbrenner N, Espeholt L, Vinyals O, Graves A, et al. Conditional image generation with pixelcnn decoders. Adv Neural Inf Process Syst. 2016;29.
106. Kolesnikov A, Lampert CH. Pixelcnn models with auxiliary variables for natural image modeling. In: International Conference on Machine Learning, 2017;1905–1914. PMLR.
107. Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. 2016. arXiv preprint arXiv:1609.02907.
108. Xu Y, Song Y, Garg S, Gong L, Shu R, Grover A, Ermon S. Anytime sampling for autoregressive models via ordered autoencoding. 2021. arXiv preprint arXiv:2102.11495.
109. Dan Y, Zhao Y, Li X, Li S, Hu M, Hu J. Generative adversarial networks (gan) based efficient sampling of chemical composition space for inverse design of inorganic materials. Npj Comput Mater. 2020;6(1):84.
110. Salimans T, Ho J. Progressive distillation for fast sampling of diffusion models. 2022. arXiv preprint arXiv:2202.00512.
111. Kellison T. An overview of sustainable development goal 11. The Routledge handbook of sport and sustainable development, 2022;261–275.
112. Louman B, Keenan RJ, Kleinschmit D, Atmadja S, Sitoe AA, Nhantumbo I, Camino Velozo R, Morales JP. Sdg 13: Climate action-impacts on forests and people. Sustainable development goals: their impacts on forests and people, 2019;419–444.
113. Zinkevich M, Weimer M, Li L, Smola A. Parallelized stochastic gradient descent. Advances in neural information processing systems. 2010;23.
114. Zhang Z. Improved adam optimizer for deep neural networks. In: 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), 2018;1–2. Ieee.
115. He S, Li Z, Tang Y, Liao Z, Li F, Lim S-J. Parameters compressing in deep learning. Comput Mater Contin. 2020;62(1):321–36.
116. Huang T, Zhang Y, Zheng M, You S, Wang F, Qian C, Xu C. Knowledge diffusion for distillation. Adv Neural Inf Process Syst. 2024;36.
117. Li M, Lin J, Meng C, Ermon S, Han S, Zhu J-Y. Efficient spatially sparse inference for conditional gans and diffusion models. Adv Neural Inf Process Syst. 2022;35:28858–73.
118. Park J, No A. Prune your model before distill it. In: European Conference on Computer Vision, 2022;120–136.
119. Zhan F, Yu Y, Wu R, Zhang J, Cui K, Zhang C, Lu S. Autoregressive image synthesis with integrated quantization. In: European Conference on Computer Vision, 2022;110–127. Springer.
120. Botelho S, Joshi A, Khara B, Rao V, Sarkar S, Hegde C, Adavani S, Ganapathysubramanian B. Deep generative models that solve pdes: Distributed computing for training large data-free models. In: 2020 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC) and Workshop on Artificial Intelligence and Machine Learning for Scientific Applications (AI4S), 2020;50–63. IEEE.
121. Grimmer J. An introduction to bayesian inference via variational approximations. Polit Anal. 2011;19(1):32–47.
122. Singh S, Wick M, McCallum A. Monte carlo mcmc: Efficient inference by approximate sampling. In: Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, 2012;1104–1113.
123. PeerJ: Offloading the computational complexity of transfer learning. PeerJ 2024.
124. DLReview: Optimization algorithms and heuristics in deep learning. Deep Learn Rev 2023.
125. Akay B, Karaboga D, Akay R. A comprehensive survey on optimizing deep learning models by metaheuristics. Artif Intell Rev, 2022;1–66
126. Chen H, Wang Y, Shu H, Wen C, Xu C, Shi B, Xu C, Xu C. Distilling portable generative adversarial networks for image translation. In: Proceedings of the AAAI Conference on Artificial Intelligence, 2020;34:3585–3592.
127. Yamazaki HV. On Depth and Complexity of Generative Adversarial Networks 2017.
128. Karnewar A, Wang O. Msg-gan: Multi-scale gradients for generative adversarial networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020;7799–7808.
129. Martinez E, Jacome R, Hernandez-Rojas A, Arguello H. Ldgan: Low-dimensional generative adversarial network for spectral image generation with variance regularization. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023;265–275.
130. Kenfack PJ, Sabbagh K, Rivera AR, Khan A. Repfair-gan: Mitigating representation bias in gans using gradient clipping. 2022. arXiv:abs/2207.10653.
131. Bartz-Beielstein T, Branke J, Mehnen J, Mersmann O. Evolutionary algorithms. Wiley Interdiscip Rev Data Min Knowl Dis. 2014;4(3):178–95.
132. Frazier PI. Bayesian optimization. In: Recent Advances in Optimization and Modeling of Contemporary Problems, 2018;255–278. Informs.
133. Nusrat I, Jang S-B. A comparison of regularization techniques in deep neural networks. Symmetry. 2018;10(11):648.
134. Nakamura K, Derbel B, Won K-J, Hong B-W. Learning-rate annealing methods for deep neural networks. Electronics. 2021;10(16):2029.
135. Hussain Z, Gimenez F, Yi D, Rubin D. Differential data augmentation techniques for medical imaging classification tasks. In: AMIA Annual Symposium Proceedings, 2017;2017:979.
136. Stiny G. Introduction to shape and shape grammars. Environ Plan. 1980;7(3):343–51.
137. Suzuki M. Improving bi-directional generation between different modalities with variational autoencoders. 2018. arXiv preprint arXiv:1801.08702.
138. Soydaner DJNC, Zhu H, Xie C, Fei Y, Tao HJE. Visual attention mechanism in deep learning. Neural Comput Appl. 2022;34:13371–85. https://doi.org/10.1007/s00521-022-05567-8.