Natural language generation and deep learning for intelligent building codes

Ruichuan Zhang[a]; and Nora El-Gohary[b]

[a] Graduate Student, Department of Civil and Environmental Engineering, University of Illinois at Urbana-Champaign, 205 N. Mathews Ave., Urbana, IL 61801, United States. E-mail: rzhang65@illinois.edu.

[b] Associate Professor, Department of Civil and Environmental Engineering, University of Illinois at Urbana-Champaign, 205 N. Mathews Ave., Urbana, IL 61801, United States (corresponding author). E-mail: gohary@illinois.edu; Tel: +1-217-333-6620.

Abstract

Many existing ACC systems require the processes of extracting regulatory information from natural language building-code requirements and transforming the extracted information into computer-processable semantic representations. These processes could, however, be jeopardized by the ambiguous nature of the natural language and the hierarchically complex structures of building-code requirements. To address this problem, this paper proposes the concept of intelligent building code for bypassing the error-prone information extraction and transformation processes. In the proposed intelligent code, the natural-language requirements in the code are connected with highly structured computer-understandable semantic information, which is represented in the form of semantic requirement hierarchies and can be readily used by computers for ACC. The paper also proposes a deep learning-based method to automatically generate such intelligent code. The method leverages the requirement hierarchy representation, a proposed deep learning unit-to-text model for generating requirement sentence segments, and a proposed semantic correspondence score for configuring the segments into requirement sentences. The method was implemented and tested on a dataset from multiple regulatory documents. The generated intelligent requirements were evaluated in terms of both natural-language requirement comprehensibility and correspondence between the natural language and the semantic representation, with the results indicating high performance for the proposed representation and method. The proposed intelligent code will help reduce ACC errors, improve requirement comprehensibility, and facilitate intelligent code analytics.

Keywords: Intelligent building code; Natural language generation; Deep learning; Automated compliance checking; Requirement representation.

1 Introduction

Building designs are governed by a variety of regulatory documents in the architecture, engineering, and construction (AEC) domain. Traditional, manual checking of the compliance of building designs with these regulatory documents is time- and cost-consuming, and prone to errors. To improve the time- and cost-efficiency and to minimize the errors of the compliance checking processes, many automated compliance checking (ACC) methods and systems have been developed. Existing ACC systems, regardless of their level of automation (e.g., semi- or full-automation), all require the extraction of regulatory information (e.g., compliance checking attribute and quantity value) from the natural-language building-code requirements and the transformation of the extracted information into computer-processable semantic representations. For example, the users of Solibri Office [1], a type of semi-automated ACC system, first read the requirements, identify the correct rule templates for the requirements, and manually extract the values for the parameters of the templates from the requirements. The state-of-the-art, rule-based fully-automated ACC systems use semantic natural language processing (NLP) rules based on semantic and syntactic features to extract semantic information elements from regulatory documents and transform them into logic forms [2]. Despite the performance achieved by the existing ACC systems, the information extraction and transformation processes within these systems could, however, be jeopardized by the ambiguous nature of the natural language and the complex and recursive semantic and syntactic structures of building-code requirements [3,4]. For example, in the existing ACC systems that employ NLP-based information extraction and transformation methods, errors resulting from these methods could further cause errors in downstream ACC processes, such as computer-processible rule-based compliance reasoning, and, eventually, errors in the final compliance checking results [5].

Aiming to bypass the error-prone information extraction and transformation processes and make building-code requirements directly computer-processable, several research efforts have been undertaken to develop computer-processable semantic representations that enable the representation of building-code

requirements directly with only limited or even zero use of natural language. Following these approaches, to create new requirements, building code developers no longer need to write natural-language requirements; instead, they would solely provide the semantic regulatory information that defines the new requirements directly in the form of such representation (i.e., aiming to eliminate the natural-language form). For example, the conceptual graphs [6] and the visual code checking language [7] were proposed to represent rules that define building-code requirements, which consist of AEC domain-specific semantic concepts (e.g., building objects and relations between these objects) and connections (e.g., conjunctions, disjunctions, and constraints), in graph-like structures (e.g., nodes and edges). These semantic concepts and connections are provided by the users of these two representations as input in the form of nodes and edges to develop graphs that represent requirements. However, these computer-processable semantic representations suffer from two limitations. First, without the corresponding natural-language requirements, it is often difficult for practitioners to understand and comprehend the regulatory information in these representations [8]. Second, even when supplemented with the corresponding natural-language requirements, these representations still lack comprehensibility because the direct correspondence/link between the two forms (i.e., the natural language and the computer-processable semantic representation) is missing, limiting their use in ACC processes.

An intelligent code, where highly structured computer-understandable semantic information that can be used directly by computers for semantic analysis tasks [9] is connected with its natural-language counterpart, takes the best of both worlds of the semantic representations and the natural language. Such intelligent code would (1) reduce the ambiguity of the natural-language requirements, while preserving its comprehensibility, (2) be both directly processable by computers and understood by humans [10], and (3) maintain the correspondence between the semantic information and the natural-language sentences.

Recent advances in data-to-text generation (e.g., [11]) in the domains of computer science and natural language generation (NLG) provide an unprecedented opportunity for developing such intelligent code for both reduced ACC errors and improved requirement comprehensibility. NLG aims to produce human-

readable text based on structured data/information or some intermediate semantic representation. Thus, there is a need for semantic NLG-based methods for converting and generating intelligent building codes in the AEC domain, i.e., methods to automatically generate natural-language sentences based on semantic regulatory information input and retain the correspondence between the words, phrases, and clauses in the sentence and their semantic sources during the generation process. However, such generation is not easy because of the following challenges: first, defining a representation that is both (1) readily semantic and computer-understandable (and thus could connect to exiting computer-processible representations used for compliance reasoning) and (2) directly linked to an NLG model (and thus could support fully automated conversion of the structured input into natural language sentences); second, generating meaningful sentences that are easy to understand; and third, retaining fidelity of the input data (e.g., by tracing back to the input data used for generating the output text) [12-14].

To address this need, first, the paper proposes a new semantic representation of building-code requirements, the multi-form semantic (MFS) requirement hierarchy and the intelligent code representation, for supporting intelligent code generation and thus downstream ACC processes, such as compliance reasoning. The hierarchy consists of simple, manageable requirement units that are semantically represented and linked, where each unit is composed of semantic information elements (SIEs) that define the requirements (e.g., subject, compliance checking attribute, and quantity value). It is represented in two supplementary forms, the surface form and the background form. The intelligent code representation consists of the natural-language requirement, its corresponding MFS requirement hierarchy, and the semantic links that indicate the correspondence between the two. Second, a deep learning and semantic NLG-based method is proposed for generating such intelligent code. The method (1) uses the MFS requirement hierarchy representation as the input form for generating the corresponding natural language requirements; (2) trains a deep learning requirement unit-to-text model to generate natural-language requirement sentence segments, using data prepared based on a large, multi-topic building-code corpus; and (3) connects these segments into intelligent requirement sentences. The natural language

104 requirements, along with their semantic counterparts (i.e., the MFS requirement hierarchies), form the
105 intelligent code.

106 ## 2 Background

107 ### *2.1 Semantic representations of natural-language requirements for automated compliance checking*

108 Existing ACC systems represent the natural-language building-code requirements in computer-
109 processable semantic representations for supporting downstream ACC tasks, such as matching regulatory
110 information to design information in building information models (BIM) and compliance reasoning. For
111 example, Garrett and Hakim [15] developed object-oriented representation schemes of requirements.
112 Ozkaya and Akin [16] proposed a design framework to incorporate requirements into digital designs.
113 Yurchyshyna and Zarli [17] used the SPARQL Query Language for RDF to represent requirements for
114 retrieving design information from BIM-based design files in the industry foundation classes (IFC)
115 format. Pauwels et al. [18] used semantic web technologies to represent requirements as a directed,
116 labeled graph of semantic concepts and connections and logic-based rules. Zhang and El-Gohary [5]
117 proposed the semantic information elements (SIEs) for representing requirements in the form of logic
118 rules that incorporate these SIEs, where the SIEs were extracted using ontologies and natural language
119 processing methods. Lee et al. [19] designed the building environment and analysis language to represent
120 objects and relations in the requirements in an object-oriented manner. Uhm et al. [20] adopted a context-
121 free grammar-based method for developing computer-interpretable rules. Dimyadi et al. [21] adopted the
122 process model and notation (BPMN)-based compliant design procedures (CDP) for describing
123 compliance checking workflows and embedding regulatory knowledge using regulatory knowledge query
124 language (RKQL). The International Code Council and AEC3's SmartCode project [22] used the
125 requirement, application, selection, and exception (RASE) markups [23] to facilitate developing
126 computer-processable rules that represent the requirements.

These semantic representations are at the core of existing ACC systems and/or software – they enable representing, processing, and checking building-code requirements automatically by computers; however, existing ACC systems typically lack the mechanisms to support fully automated (i.e., without human annotations or manually crafted extraction/transformation rules) extraction and conversion of natural-language requirements into these representations. For example, Weise et al. [22] requires manual annotation of the codes with the RASE tags. Zhang and El-Gohary [5] although can do the information extraction fully automatically, still requires hand-crafted extraction and transformation rules to extract the semantic information from the natural language and convert the extracted information into computable logic rules. Recent research efforts have been undertaken to develop flexible and highly automated methods to support such extraction and conversion, by leveraging artificial intelligence (AI) technologies, such as machine learning and NLP.

2.2 Deep learning for text analytics

Deep learning methods use deep neural networks that consist of stacks of layers to capture different levels of information representations from data [24]. Deep learning methods have drastically improved the state-of-the-art performance in automatically processing and understanding natural-language data, and meanwhile reduced or eliminated the manual effort in feature engineering compared to traditional machine learning methods. Recurrent neural networks (RNN) are deep learning models consisting of internal states specifically designed to process sequential data, such as text data, which consist of sequences of words. To solve the problem of vanishing gradient and improve the capability to capture long-term semantic and syntactic dependencies, two variants of the RNN, long short-term memories (LSTM) and gated recurrent units have been proposed and used. RNN-based models have been widely used in natural language processing, understanding, and generation tasks, such as text classification (e.g., [25]), sequence labeling (e.g., [26]), semantic parsing (e.g., [27]), and machine translation (e.g., [28]). A limited number of research efforts have been undertaken on deep learning-based methods for solving text analytics tasks in the AEC domain. For example, Pan and Zhang [29] developed RNN-based models to

analyze building information modeling (BIM) log data for extracting and discovering knowledge that supports design decisions. Zhong et al. [30] used bidirectional LSTM and CRF models to extract procedural constraints from construction regulations. Zhang and El-Gohary [31] used bidirectional LSTM and CRF models with transfer learning strategies for extracting semantic and syntactic information elements from building-code sentences.

2.3 Data-to-text natural language generation

Natural language generation (NLG) is the process of representing the semantic information contained in the input data – which could be in various forms such as tables, images, or formal languages – in the form of natural language for the purpose of information digestion and communication [32]. NLG plays an important role in intelligent systems such as spoken dialogue systems (e.g., [33]), image captioning (e.g., [34]), text summarization (e.g., [35]), and programming code management (e.g., [36]). Data-to-text generation is the NLG process that automatically generates text from non-linguistic, structured input, such as records in databases and knowledge bases [37].

Data-to-text generation methods range from template- and rule-based methods to machine learning-based methods, including deep learning-based methods. Template- and rule-based methods rely on manually developed templates for forming sentences and rules to fill in the templates. They require manual effort for developing and maintaining the templates and rules, and typically lack the flexibility to deal with complex text [32]. Machine learning-based methods, instead of relying on predesigned rules and templates, use machine learning models to automatically capture the semantic and syntactic patterns in the training data, which are later used for supporting the generation of new text. Different deep learning model architectures were proposed for NLG applications, such as RNN-based encoder-decoder architectures (e.g., [11,38-41]), transformer-based architectures (e.g., [42]), and variational autoencoders (e.g., [43]). Variants of these model architectures have also been proposed. For example, the attention mechanism (e.g., [11,41]) and copying mechanism (e.g., [38,40]) in the encoder-decoder architecture

were proposed to improve the ability of the deep learning models to capture the structural dependency and maintain the data fidelity [39].

3 State of the art and knowledge gaps in regulatory text generation

Existing methods and systems for generating regulatory text (e.g., requirements or documents) in the AEC domain are mainly based on premade templates and rules. For example, Thewalt and Moskowitz [44] used a network of decision tables for representing design standards and a set of handcrafted templates for standard text generation. Ryoo et al. [45] proposed a web-based construction specification system where users can query existing specifications, guidelines, and other materials for drafting new specifications. Commercial software such as e-Specs [46] and BIMdrive [47] use premade templates (e.g., templates based on the National Master Specifications) to facilitate the development and maintenance of specifications. The latest IDM schema [48] specified how IDM documents are developed, exchanged, and shared in the idmXML format and the IDM generation rules. These efforts have provided important insights and practical value for generating regulatory text in the AEC domain. Despite their importance, they typically have three limitations. First, developing and updating the templates and rules for generating regulatory text is labor-intensive and time-consuming. A comparative analysis of hybrid methods (that include machine learning) and entirely rule-based methods for data-to-text generation in the literature has demonstrated significant time and resource savings when leveraging machine learning [49]. For generating building-code requirements, this time and cost could significantly grow because a large number of building codes exist, including amendments to accommodate local needs, and many of these codes/amendments are subject to frequent and regular updates. Second, such manually developed templates and rules are typically rigid and hard to scale up in real-world applications and across different types of text. Methods using templates and rules are generally outperformed by machine learning-based methods in various data-to-text generation tasks (e.g., [32,50]). Flexibility and scalability are especially important for generating building-code requirements, because different types of building codes usually have different syntactic and semantic characteristics. Testing and adapting the templates and rules to

accommodate the variabilities across the different codes and requirements would be practically difficult. Third, they are limited in capturing the semantic and syntactic complexities and variations in the building-code requirements. For example, energy code requirements tend to have hierarchically complex semantic and syntactic structures including deeply nested clauses, conjunctive and alternative obligations, and multiple restrictions and exceptions [3]. Developing rules to deal with such structural complexity based on limited templates (e.g., the templates are usually limited in the patterns of semantic features and structures they represent) is challenging.

Deep learning-based methods, on the other hand, have achieved state-of-the-art performance in many NLG tasks including data-to-text tasks (e.g., [38-40]). However, there is a lack of efforts that used deep learning-based NLG methods for AEC applications. This poses a missed opportunity, particularly for intelligent code generation efforts.

4 Research methodology

The research methodology was composed of the following primary research tasks:

- Representation development: In developing the representation (see Section 5), to overcome the challenges outlined in Section 1, a number of criteria were defined. These included developing a representation that: (1) is readily semantic and computer-understandable, while being directly linked to the generation model; (2) supports the generation of meaningful sentences that are well understandable to humans; and (3) retains fidelity of the input data (i.e., keeping the link between the semantic representation and the natural language sentences). The representation development process included the following steps: (1) define the basic elements and the structure to represent the semantics; (2) define the different forms to represent the semantics and to serve as the input to the code generation model; and (3) define the links between these semantic elements and the forms.
- Method development: The methodology for developing the semantic NLG-and deep learning-based method for generating intelligent code (see Section 6) included three primary steps:

- Data preparation: Datasets for training and testing were prepared, as discussed in more detail in Section 6.1.
- Text generation model development: A deep learning-based generation model was developed for generating natural-language text (sentence segments) corresponding to the structured input representation, which included model design and training. The model design process included two primary steps: (1) select the learning model and adapt it to the problem at hand (i.e., automatically generating requirement sentence segments given the semantic information); and (2) define the design practices that are needed to further optimize the model for the task at hand. The RNN-based encoder-decoder model [51] was selected and adapted and three practices were defined (LSTM, bidirectional architecture, and attention mechanism), as discussed in Section 6.2.1. For model training, four training practices were defined, as discussed in Section 6.2.2.
- Intelligent building code generation method development: A divide-and-conquer paradigm was selected and followed for developing the method (see Section 6.3), where requirement segments (instead of the whole requirement sentence) were generated (using the trained text generation model), linked to the semantic representation, and connected into intelligent code requirements.

- Experimental results and analysis: A set of experiments were conducted to test and evaluate the effectiveness of the proposed representation and the performance of the proposed intelligent code generation method, including flexibility across different types of codes and standards and sentence computability levels (see Section 7).

5 Proposed representation: requirement hierarchy and intelligent code

5.1 Multi-form semantic requirement hierarchy representation

The paper proposes a new semantic representation, the multi-form semantic (MFS) requirement hierarchy [see Fig. 1(a) and (b)], to model building-code requirements, especially the hierarchically complex requirements with restrictions and exceptions, for facilitating the generation of intelligent code, which could further support downstream ACC processes, such as compliance reasoning based on semantic

representations (e.g., logic). An MFS requirement hierarchy models a requirement in a hierarchical structure that consists of several requirement units. Each requirement unit consists of several SIEs (as listed in Table 1), which are the constituent concepts (e.g., subject and compliance checking attribute), relations (e.g., subject relations), and indicators (e.g., deontic operator indicator) that define a requirement or a condition in a requirement. Each unit has at least one subject or compliance checking attribute and may or may not have other SIEs. Each unit does not have any secondary SIEs such as restrictions and exceptions, and thus is easily processable using most of the existing semi-automated or automated compliance checking methods and systems. There are two types of relations between the requirement units: simple and complex relations. Simple relations include conjunctions (e.g., "and") and disjunctions (e.g., "or"). Complex relations include exceptions and restrictions.

Table 1. Semantic Information Elements (SIEs) for Defining Requirements for Compliance Checking [1].

Semantic information element	Definition
Subject	An ontology concept representing a thing (e.g., building element) that is subject to a particular requirement
Subject relation	A term or phrase that defines the type of relationship between two subjects, a subject and an attribute, or a subject or an attribute and a quantity (e.g., "equipped")
Compliance checking attribute	An ontology concept representing a specific characteristic of a "subject" that is checked for compliance (e.g., "width")
Deontic operator indicator	A term/phrase that indicates the deontic type of the requirement (i.e., obligation, permission, or prohibition)
Comparative relation	A term/phrase for comparing quantitative values, including "greater than or equal to," "greater than," "less than or equal to," "less than," and "equal to"
Quantity value	A numerical value that defines the quantity
Quantity unit	The unit of measure for a "quantity value"
Reference	A term or phrase that denotes the mentioning or reference to a chapter, section, document, table, or equation in a building-code sentence

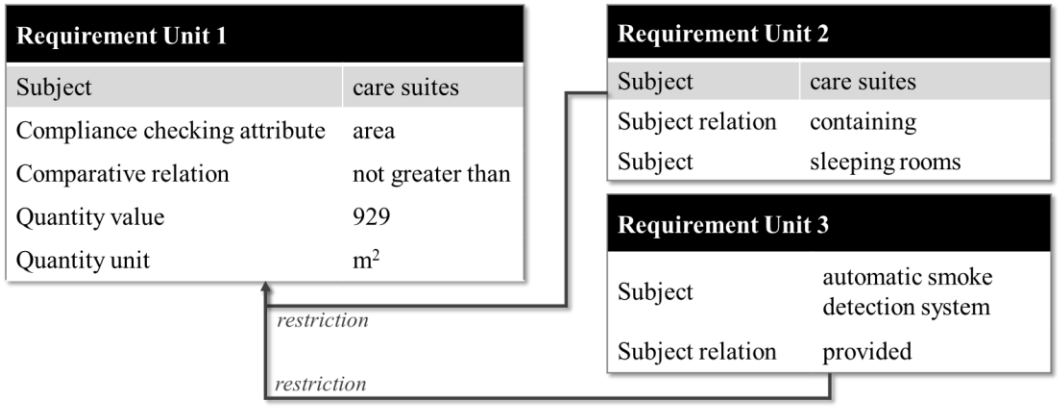
The MFS requirement hierarchy is represented in two supplementary forms: the surface form [see Fig. 1(a)] and the background form [see Fig. 1(b)]. The surface form shows the requirement units and the SIEs within each unit in a hybrid, templatized and graphical, format. Similar to the conceptual graph [6] and the visual code checking language [7], the surface form can be visualized and manipulated for the purpose of requirement editing, development, or navigation. For example, the surface form can be embedded as a part of a user interface for the requirement developers to add or modify the requirement units, the

restriction dependencies between the units, and the SIEs within each unit, for the purposes of generating natural language requirements that are readily semantic and computer-understandable (i.e., the intelligent code). The background form shows the SIEs and the predicate-argument structures between the SIEs and the requirement units in a sequential format. The background form consists of two types of tokens: semantic tokens to represent the semantic meanings of the SIEs (e.g., words and numbers), and syntactic tokens to indicate the predicate-argument structures (e.g., parenthesis and vertical bars). The form can be directly embedded using vector representations and thus fed into the deep learning requirement unit-to-text generation model for generating the requirement sentence segments corresponding to the units in the MFS requirement hierarchy. The surface form can be converted into the background form using rules (e.g., rules to concatenate words from the surface form and add syntactic tokens to build the background form). Examples of these rules, which were developed in Python 3, are shown in Fig. 1. Due to its semantic and structured representation, the MFS requirement hierarchy is potentially compatible and integrable with other ACC representations and workflows (e.g., CDP).

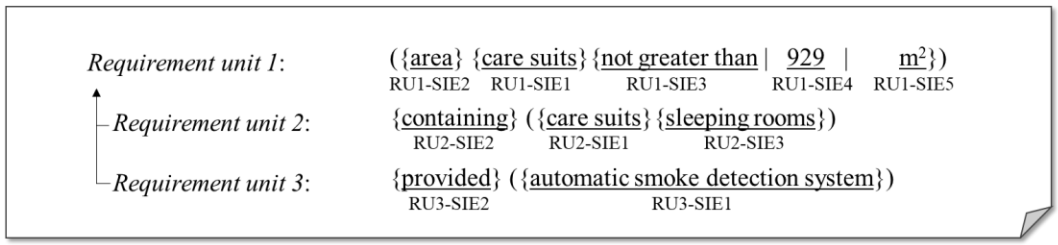
5.2 Intelligent code representation

The paper defines an intelligent code as a natural-language requirement connected with its corresponding MFS requirement hierarchy. It consists of three parts – the natural-language requirement, its corresponding MFS requirement hierarchy, and the semantic links that indicate the correspondence between the surface form and the background form of the hierarchy and the natural-language requirement (an example to illustrate the representation of the intelligent requirement is shown in Fig. 1). Each semantic correspondence key consists of a requirement unit identifier and an SIE identifier, and every token in the natural-language requirement and the requirement hierarchy is annotated with such a semantic correspondence key. Thus, intelligent requirements can be understood by both humans (via the natural-language requirement) and computers (via the MFS requirement hierarchy), and meanwhile, preserve the semantic links between the natural-language requirement and the requirement units and SIEs in the hierarchy (via the semantic correspondence keys).

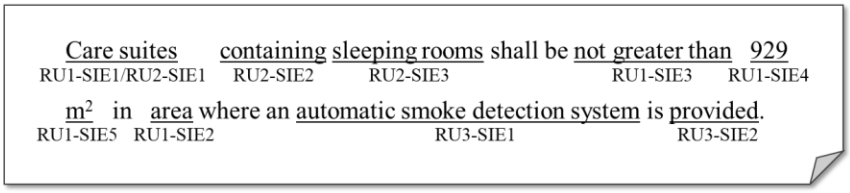
Fig. 1 shows an example to illustrate the concept of intelligent requirements (using a sentence from IBC 2018 [52]). The MFS requirement hierarchy is shown in Fig. 1(a) (the surface form) and Fig. 1(b) (the background form). The natural-language requirement with the semantic correspondence keys is shown in Fig. 1(c). In this example, the whole requirement is modeled as a requirement hierarchy consisting of three requirement units, with units 2 and 3 being restrictions of unit 1, which is the main requirement unit in this hierarchy.



(a) Multi-form semantic requirement hierarchy representation (surface form)



(b) Multi-form semantic requirement hierarchy requirement hierarchy representation (background form) with semantic correspondence keys



(c) Natural-language requirement with semantic correspondence keys

1. RU = requirement unit; SIE = semantic information element
2. Syntactic tokens (in the background form): () = all arguments; {} = an argument or predicate; | = segmentation of semantic information elements within an argument
3. Example rules for converting the surface form to the background form: i. SIE-to-predicate rule: the subject relation serves as the predicate and is capsulated by a pair of curly brackets ("{}"); and ii. SIE-to-argument rule: the semantic information elements, other than the subject relation, serve as the arguments, each capsulated by a pair of curly brackets ("{}"); all the arguments are capsulated by a pair of round brackets ("()"); and different SIEs in an argument are segmented by vertical bars ("|").

Fig. 1. Example semantically annotated requirement: (a) surface form of the requirement hierarchy; (b) background form of the requirement hierarchy with semantic correspondence keys; (c) natural-language requirement sentence with semantic correspondence keys.

6 Semantic NLG- and deep learning-based method for generating intelligent code

The method for intelligent code generation was proposed and implemented on corpora of building-code requirements. The methodology consists of three primary steps, as per Fig. 2: data preparation, deep learning-based requirement unit-to-text generation model development, intelligent building-code

requirement generation. An example to illustrate how the intelligent requirements are generated (Step 3), using the trained requirement unit-to-text model (Step 2), is shown in Fig. 3.

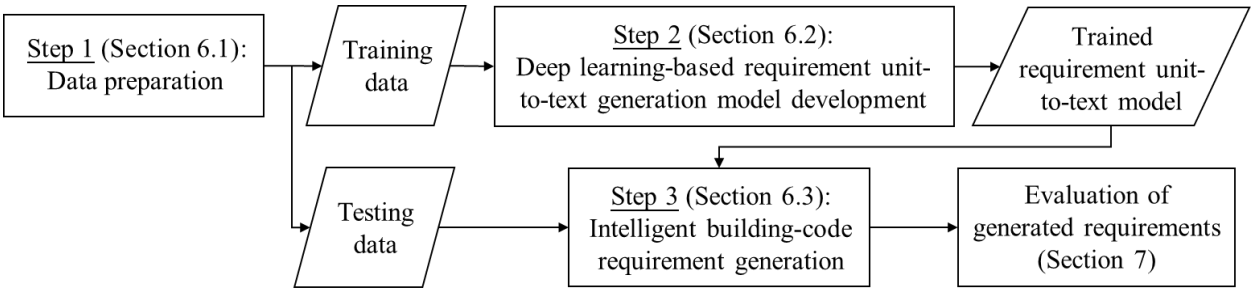


Fig. 2. Research methodology for NLG- and deep learning-based intelligent building code generation.

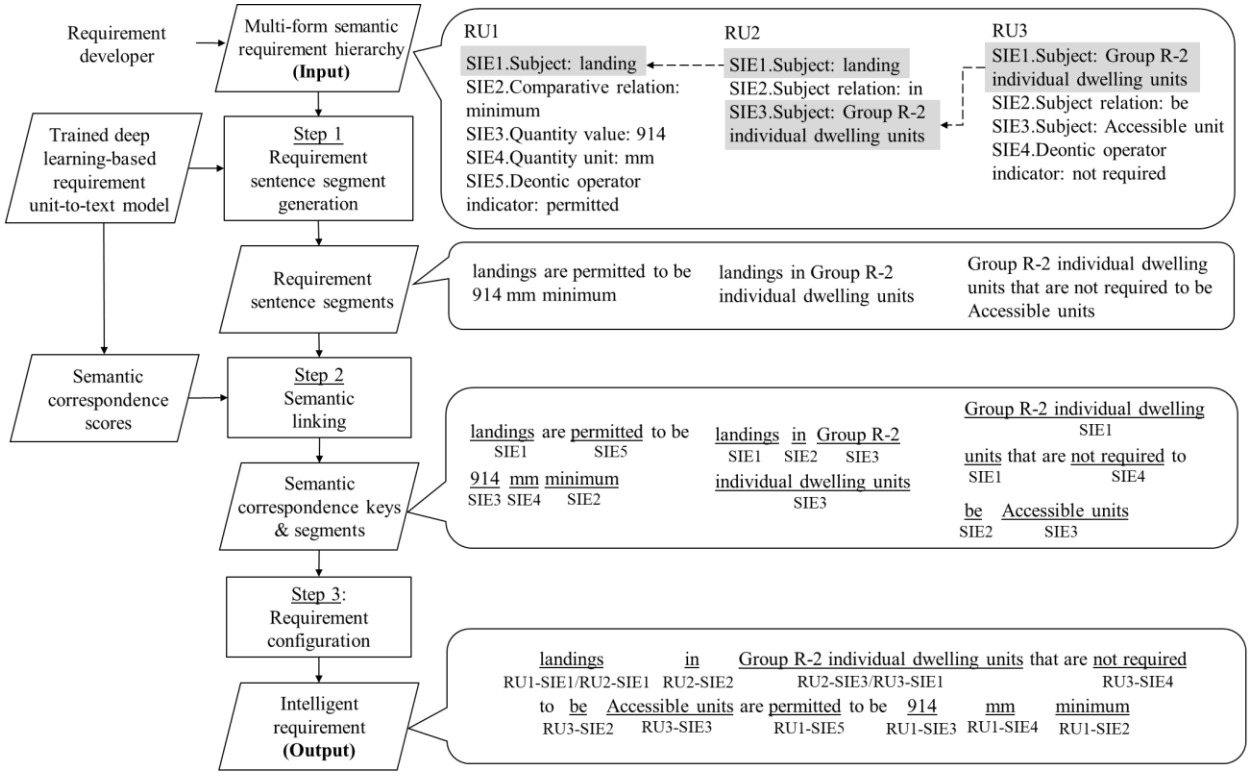


Fig. 3. Intelligent building code generation.

6.1 Data preparation

The training and testing data were prepared for training the requirement unit-to-text generation model and evaluating the requirement generation results. The data are semantically annotated building-code requirements, including the natural-language requirement and the corresponding MFS requirement

hierarchy with both its surface and background forms. Both the training and testing data were prepared following three steps: corpus preparation, sentence selection, and sentence annotation.

6.1.1 Corpus preparation

Two building-code corpora were prepared – the training and testing corpora. The training corpus consists of sentences from the International Building Code (IBC) and its amendments. The testing corpus consists of sentences from three types of regulatory documents, including the IBC, International Energy Conservation Code (IECC), and Americans with Disabilities Act Standards for Accessible Design (ADA Standards). The text files of these documents were crawled from webpages or converted from PDF files, and they were preprocessed following three steps: sentence segmentation, tokenization, and pruning. Sentence segmentation aims to detect the sentence boundaries (e.g., punctuations) and segment the text into sentences. Sentence tokenization aims to further split the sentences into tokens (e.g., words). Sentence pruning aims to remove the sentences or sentence segments that are not requirements (e.g., headings). The Natural Language Toolkit (NLTK) in Python was used for text preprocessing.

6.1.2 Sentence selection

Two groups of sentences and sentence segments were selected from the two corpora, respectively. The training group consists of about 7,500 sentences and sentence segments including about 100,000 tokens (e.g., words, numbers, and punctuations); and the testing group consists of about 600 sentences and fragments including about 15,000 tokens. For evaluation purposes, sentences having different levels of computability (i.e., the ability of sentences to be automatically processed, represented, and checked by ACC systems) were selected, based on the computability definition and metrics in [53].

6.1.3 Data annotation

The two sentence groups were annotated and converted into requirement hierarchies. The training group was automatically annotated using the pretrained requirement hierarchy extraction model [54] and the deep information extraction model [31]. The testing group was manually converted into requirement hierarchies and annotated with the semantic information elements by four experts – two from academia

(faculty) and two from industry – forming the gold standard for evaluation. A purposive sampling strategy, which pinpoints a specific type of participants according to predefined selection criteria [55], was adopted for selecting the experts. Three main selection criteria were defined: (1) expertise in the AEC domain; (2) familiarity with building codes and compliance checking processes; and (3) awareness of natural language processing and text analytics techniques. Each expert independently annotated the entire set of selected sentences, with an initial inter-annotator agreement of 80% in F1 measure, which indicates good reliability of the annotations [56]. The discrepancies among the annotations were then resolved by the experts to reach full agreement on the final annotations.

6.2 Deep learning-based requirement unit-to-text generation model development

6.2.1 Model design

The RNN-based encoder-decoder model [51] was adapted to automatically generate the corresponding natural-language sentence segment, given the background form of a requirement unit. A sentence segment is defined as the natural-language counterpart of a requirement unit and is part of the building-code sentence corresponding to the whole requirement hierarchy that contains the unit. For example, in Fig. 3, "landings are permitted to be 914 mm minimum" is the sentence segment corresponding to the requirement unit with a subject of "landings", a quantity value of "914", a quantity unit of "millimeter", and a comparative relation of ">=". The model was selected because it has achieved state-of-the-art performance in various data-to-text generation tasks and has the potential to be adapted to the requirement unit-to-text generation task at hand.

The model structure consists of two main parts: the encoder and the decoder, each consisting of several RNN layers, as illustrated in Fig. 4. The encoder [as per Eq.(1)] transforms the input sequence $[x_1, x_2, \dots, x_l]$ [i.e., the sequence of tokens (words and syntactic symbols) in the background form of the input requirement unit] of length l into a context vector representation c, which captures the semantic and syntactic information of the entire input sequence [as per Eq.(3)], based on the attention mechanism and the alignment weights α [57] [as per Eq.(2)], which captures the correlations between the tokens in the

input and output sequences. The decoder [as per Eq.(4)] further transforms the vector representation generated by the encoder into the output sequence $[y_1, y_2, \ldots, y_k]$ [i.e., the sequence of tokens (words, numbers, and punctuations) in the output requirement sentence segment] of length *k*. Each RNN layer consists of several stacked RNN units – each computes feature representations based on the input information corresponding to the current state, and the information propagated from the last state so that the information from previous states could be captured. In Eqs.(1)-(5), h_i is the RNN state corresponding to position *i* in the input sequence, s_o is the RNN state corresponding to position *o* in the output sequence, *f* is the RNN, and φ is a multilayer neural network.

$$h_i = f(x_i, h_{i-1}) \quad (1)$$

$$\alpha_{o,i} = \frac{e^{\varphi(s_{o-1}, h_i)}}{\sum_{i'} e^{\varphi(s_{o-1}, h_{i'})}} \quad (2)$$

$$c_o = \sum_{i=1}^{l} \alpha_{o,i} h_i \quad (3)$$

$$s_o = f(y_{o-1}, s_{o-1}, c_o) \quad (4)$$

To further optimize the model for the task at hand, three model design practices were followed. First, an RNN alternative – the LSTM [58] was used to alleviate the vanishing gradient problem during the training of the RNN-based models. Second, the bidirectional architecture was used in the encoder – both the forward and backward tokens instead of only the forward one were considered when learning the feature representations for the current token [59], in order to improve the ability of the RNN-based models to deal with long-term dependencies in the requirement sentences. Third, the attention mechanism [57] was adopted in the model to incorporate the correlation between the tokens in the input sequence and the tokens in the output sequence in an attentive read vector representation, which replaces the fixed-length vector representation, for better handling of long sentences.

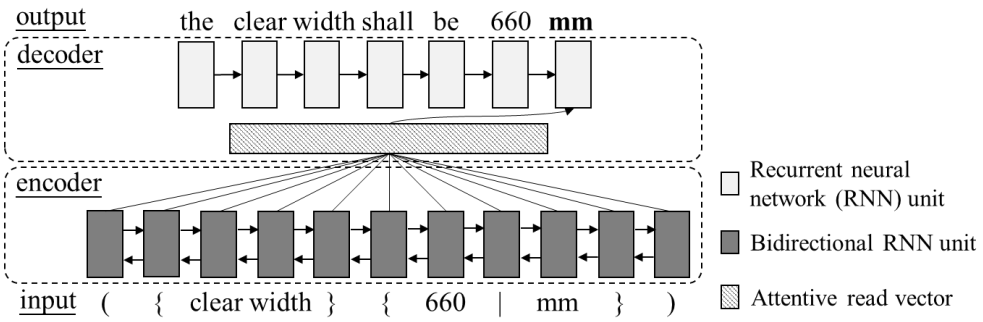


Fig. 4. Encoder-decoder model for requirement unit-to-text generation.

6.2.2 Model training

The following four training practices were followed. First, for optimizing the model parameters, perplexity was chosen as the loss function and was minimized during the training process. Perplexity is defined as the inverse probability of the training sentences given a language model (e.g., the encoder-decoder model), normalized by the number of tokens in the training sentences [60]. Second, for determining the best values of the hyperparameters for both model structure and model training, the training data was split into a 9:1 ratio for model training and validation, respectively. Third, for improving computational efficiency, the training process was stopped at 20 epochs or when the change of the value of the loss function (i.e., perplexity) between two consecutive training epochs was less than the threshold. Fourth, for improving the sentence generation performance, the model learning rate was gradually decreased during the training process [51].

6.3 Intelligent building-code requirement generation

The method for generating intelligent building-code requirements includes three steps, as illustrated in Fig. 3: requirement sentence segment generation, semantic linking, and requirement configuration.

6.3.1 Requirement sentence segment generation

For each unit in the input requirement hierarchy, a corresponding sentence segment was generated using the trained requirement unit-to-text generation model (see Step 1, Fig. 3). Given its surface form, first, a requirement unit was converted into its background form. Second, the background form was encoded into a vector representation by the encoder of the trained requirement unit-to-text generation model. Third, the

vector representation was decoded to generate the output sentence segment [which is composed of a sequence of tokens (words, numbers, and punctuations)] successively by the decoder of the model. Each time, for the decoder to generate a new token, two steps were followed: (1) a classifier g (e.g., a multilayer neural network with a softmax function) is applied to the decoder state, context vector, and the previously generated token to compute a probability distribution $p(y_o) = g(y_{o-1}, s_o, c_o)$ over the vocabulary; and (2) the beam search algorithm [61] keeps track of the best candidate sentence segments in terms of probability, until all the candidates reach the "end of a sentence" token, and returns the candidate sentence segment with the highest probability.

6.3.2 Semantic linking

This step aims to further link each token in the generated sentence segments to the SIEs in the requirement units (see Step 2, Fig. 3), using two sub-steps. First, semantic correspondence scores are calculated using the unit-to-text model. Each score measures the correspondence between a token in the sentence segment and a token in an SIE in the requirement unit according to the copying mechanism in the unit-to-text generation model [38]. The copying mechanism refers to the computational mechanism that locates the input tokens and places these tokens into the output sequence, thereby generating the output sentence [38]. As per Eq.(5), for each token t_o at position o in the generated sentence segment and each token t_i at position i in the SIEs (in the background form of the input requirement unit), the semantic correspondence score $score_{o,i}$ equals to the similarity between a linear projection (denoted by W) of the input and output RNN states (denoted by h_i and s_o, which correspond to the input background form and the output sentence segment) of the model. The hyperbolic tangent function (tanh) was adopted because compared to other types of activation functions (e.g., sigmoid and rectified linear unit), tanh typically achieves higher computational performance when used in RNN-based models [62].

$$score_{o,i} = \tanh(h_i W s_o) \quad (5)$$

Second, based on the calculated correspondence scores, for a token t_o in the generated sentence segment, the candidate semantic token t_p in the SIEs is determined by finding the argument of the maxima of the

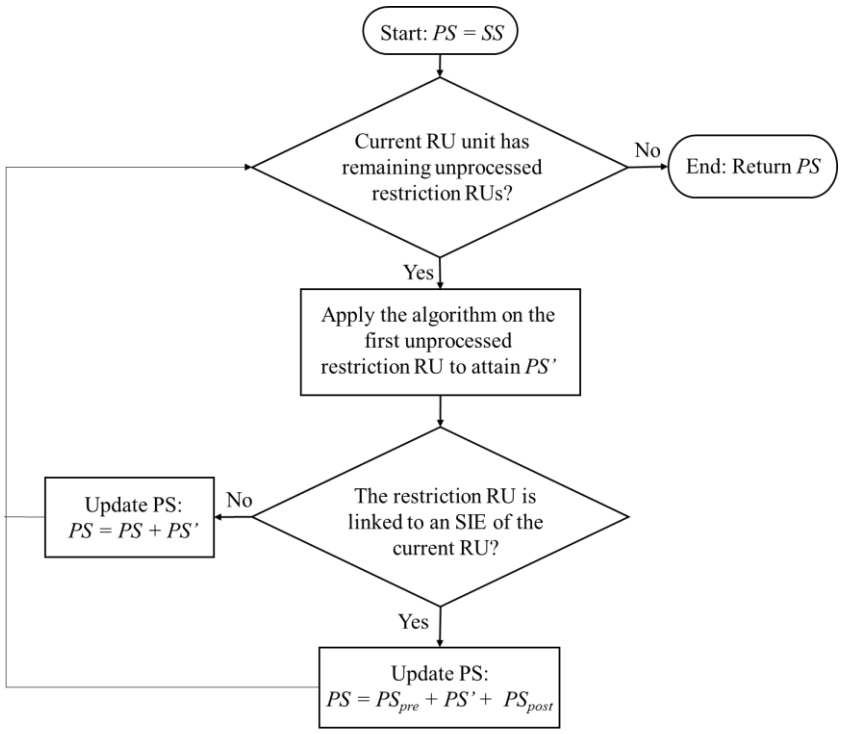
semantic correspondence scores, as per Eq.(6). The final semantic token r equals to t_p, if t_p is not a syntactic token, in which case a semantic correspondence key is generated to link t_o to the SIE. Otherwise, r is void, and t_o is not linked to any SIEs, as per Eq.(7), where S is the set of all types of syntactic tokens used in the requirement hierarchy (i.e., brackets and vertical bars). For example, in the generated sentence segment "landings are permitted to be 914 mm minimum" in Fig. 3, "landings" is linked to the first SIE "subject: landings", whereas "are" is not linked to any of the SIEs in the requirement hierarchy.

$$p = \underset{0<i\leq l}{\operatorname{argmax}}\, score_{o,i} \quad (6)$$

$$r = \begin{cases} t_p, & if\ t_p \notin S \\ \text{void}, & if\ t_p \in S \end{cases} \quad (7)$$

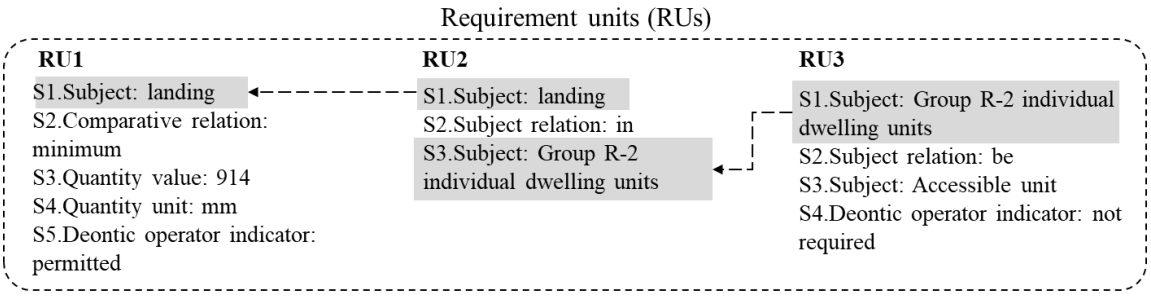
6.3.3 Requirement configuration

This step aims to combine the generated sentence segments into a whole requirement using a depth-first insertion and concatenation algorithm. Starting at the main requirement unit (i.e., the requirement unit that does not serve as a restriction to any other units), the algorithm recursively builds up the whole requirement using the sentence segments of restriction units. In each recursion (as per Fig. 5), a partial sentence (PS) of the current unit is updated by inserting the PS of the restriction unit to the PS of the current unit (when the two units share one or more SIEs) or concatenating the PSs of the current and restriction units (when the two units do not share an SIE). For example, in the requirement configuration example shown in Fig. 6, the current unit RU2 and the restriction unit RU3 share "Group R-2 individual dwelling units". Accordingly, the PS of RU3 ("Group R-2 individual dwelling units that are not required to be Accessible units") is inserted in the PS of RU2 ("landings in Group R-2 individual dwelling units"), resulting in the following updated PS: "landings in Group R-2 individual dwelling units that are not required to be Accessible units". A recursion ends when all the restriction units of the current unit are processed. The algorithm returns the PS of the main unit as the whole requirement.



RU= requirement unit; SIE=semantic information element;
SS=sentence segment corresponding to the current RU;
PS=partial sentence corresponding to the current RU;
PS'=partial sentence corresponding to the restriction RU;
PS_{pre}=the subsequence of PS that is before the linked SIE;
PS_{post} =the subsequence of PS that is after the linked SIE.

Fig. 5. Depth-first insertion and concatenation algorithm for requirement configuration.



Requirement configuration

Recursion	Current RU	Restriction RU	Shared semantic information element (SIE)	Partial sentence (PS) of current RU, start of recursion	PS of current RU, end of recursion
1	RU1 (main unit)	RU2	landings	landings are permitted to be 914 mm minimum	landings in Group R-2 individual dwelling units that are not required to be Accessible units are permitted to be 914 mm minimum
2	RU2	RU3	Group R-2 individual dwelling units	landings in Group R-2 individual dwelling units	landings in Group R-2 individual dwelling units that are not required to be Accessible units
3	RU3	-	-	Group R-2 individual dwelling units that are not required to be Accessible units	Group R-2 individual dwelling units that are not required to be Accessible units

Fig. 6. Example to illustrate requirement configuration.

7 Experimental results and analysis

Three sets of experiments were conducted to (1) optimize the deep learning-based requirement unit-to-text generation model in the proposed intelligent code generation method; (2) test if the proposed MFS requirement hierarchy representation and the semantic correspondence score with the copying mechanism are effective in improving the model's ability to generate the intelligent code; and (3) test the flexibility of the proposed method, i.e., test the comprehensibility and semantic linking of the generated intelligent requirements across different types of codes/standards and requirements with different computability levels.

In testing and evaluating the proposed method, the requirement hierarchies in the gold standard data (see Section 6.1.3) were used as the input to the proposed method. The generated requirement sentences were

then compared against the gold standard sentences and evaluated using a set of metrics (see Section 7.1). An example of the gold standard sentences, the requirement hierarchies, and the generated sentences for illustrating the evaluation process is shown in Fig. 7).

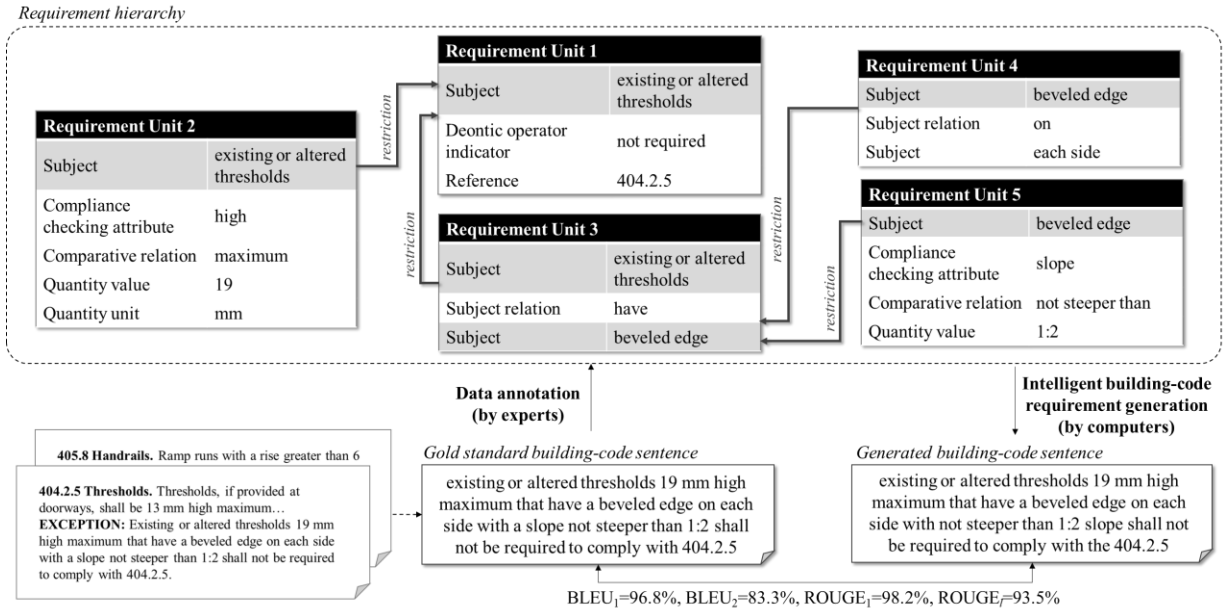


Fig. 7. Example generated requirement and its corresponding gold standard requirement.

7.1 Evaluation metrics

The generated intelligent requirements were evaluated in terms of both (1) comprehensibility of the generated natural-language requirements, and (2) semantic linking correctness of the intelligent requirement (i.e., correctness of the semantic correspondence between the natural-language requirements and the requirement hierarchy representation).

7.1.1 Evaluation of generated natural-language requirements

The generated natural-language sentences were evaluated in terms of adequacy and fluency. Based on Papineni et al. [63], a generated sentence that uses same/similar words (unigrams) as the gold standard sentence tends to satisfy adequacy; and a generated sentence that has n-gram matches with the gold standard sentence tends to satisfy fluency. Both adequacy and fluency are used together as indicators of the comprehensibility of the natural language [63]. Two metrics were used to measure adequacy and

fluency: bilingual evaluation understudy (BLEU) and recall-oriented understudy for gisting evaluation (ROUGE).

BLEU [63] measures the number of matches between the *n*-grams (continuous sequences of *n* tokens including words, numbers, and punctuations, e.g., bigram is a sequence of two adjacent tokens) in the generated sentences and the n-grams in the gold standard sentences. BLEU_N is a modified metric that measures the precision of matching using weighted average; it is the weighted average of the *n*-gram precisions p_n [e.g., weighted average of unigram precision (p_1) and bigram precision (p_2)], as per Eq.(8) [63], where w_n is the weight of p_n, *N* is the length of the longest *n*-gram in calculating *n*-gram precisions, and *b* is a brevity penalty. Here, p_n is measured using Eq.(9), where, for one generated sentence, m_n is the number of *n*-grams that are in both the generated sentence and the gold standard sentence (e.g., number of 2-gram matches) and GM_n is the total number of *n*-grams in the generated sentence (e.g., total number of 2-grams). The brevity penalty aims to penalize generated sentences that are briefer than the gold standard sentences., as per Eq.(10), where *s* is the length of the generated sentence and *r* is the length of the gold standard sentence. A high BLEU indicates that the generated sentences align well to the gold standard sentences, and thus the generated sentences have high comprehensibility (e.g., [64-66]). In this paper, BLEU_1 and BLEU_2 (i.e., $N \in \{1, 2\}$) and uniform weights (i.e., $w_1 = 1$ for BLEU_1, and $w_1 = w_2 = 0.5$ for BLEU_2) were used.

$$\text{BLEU}_N = b \exp\left(\sum_{n=1}^{N} w_n \, log \, p_n\right) \quad (8)$$

$$p_n = \frac{m_n}{GM_n} \quad (9)$$

$$b = \begin{cases} 1 & \text{if } s > r \\ e^{1-s/r} & \text{if } s \leq r \end{cases} \quad (10)$$

ROUGE [67] also measures matches between the n-grams in the generated sentences and the n-grams in the gold standard sentences. ROUGE_n specifically measures the weighted harmonic mean of the *n*-gram precision (p_n) and recall (r_n) [e.g., the harmonic mean of the unigram precision (p_1) and unigram recall (r_1)], as per Eq.(11), where β is the weighting factor. Here, r_n is measured using Eq.(12), where, for one

generated sentence, m_n is the number of n-grams that are in both the generated sentence and the gold standard sentence (e.g., number of unigram matches), and SM_n is the total number of n-grams in the gold standard sentence (e.g., total number of unigrams). A high ROUGE indicates that the generated sentences align well to the gold standard sentences, and thus the generated sentences have high comprehensibility (e.g., [11,38,68]). In this paper, ROUGE_1 and ROUGE_l (i.e., $n \in \{1, l\}$) were used, with the weighting factor $\beta = 1$ for ROUGE_1 and $\beta = p_l / r_l$ for ROUGE_l [67], where l is the longest n-gram matched between the two sentences.

$$\text{ROUGE}_n = \frac{(1+\beta^2) r_n p_n}{r_n + \beta^2 p_n} \quad (11)$$

$$r_n = \frac{m_n}{SM_n} \quad (12)$$

7.1.2 Evaluation of semantic links between natural language and semantic representation

Precision, recall, and F1 measure were used to evaluate the semantic linking [i.e., the linking of the tokens (words, numbers, and punctuations) of the generated sentence segments to the SIEs in the requirement units, as in Section 6.3.2)], as per Eqs.(13)-(15), where TP is the number of true positives (i.e., number of tokens that are correctly linked), FP is the number of false positives (i.e., number of tokens that are incorrectly linked), and FN is the number of false negatives (i.e., number of tokens that are not linked but should have been). TP, FP, and FN were calculated based on the expert annotations (see Section 6.3.1). Perfect (100%) precision, recall, and F_1 measure indicate that all generated natural-language requirements are perfectly corresponding to all input requirement hierarchies (i.e., SIEs in the requirement hierarchies).

$$\text{Precision} = \frac{TP}{TP + FP} \quad (13)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (14)$$

$$\text{F}_1 \text{ measure} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (15)$$

7.2 Hyperparameter optimization

The requirement unit-to-text generation model was trained using Tensorflow built in Python 3, and run using the Tesla K80 GPU provided in the Google Colaboratory. The values of the main hyperparameters for requirement generation are shown in Table 2. The values of the hyperparameters were determined based on: (1) the cross-validation performance of the grid search over different values [e.g., three numbers (one, two, and four) of RNN layers were tested in cross-validation, and two was selected because the model with two RNN layers achieved the best performance]; (2) the characteristics of the building-code sentences used in the experiments (e.g., the maximum length of input sentences is set as 100 based on the corpus used); or (3) the practices in the referenced papers that use RNN-based sequence-to-sequence model in data-to-text generation (e.g., the batch size for the training data was determined following [38]).

Table 2. Main Hyperparameters for Requirement Generation

Hyperparameter	Value
Model structure hyperparameters	
Number of recurrent neural network (RNN) layers in each of the encoder and decoder	2
Number of RNN units in each layer	512
Type of RNN unit	Long short term memory (LSTM)
Maximum length of requirement sentence	100
Model training hyperparameters	
Recurrent dropout rate	0.2
Batch size for training data	32
Size of gradient normalization	5

The model structure was optimized. For example, different numbers of RNN layers and units in each layer were tested. Three numbers of RNN layers were tested: one (shallow), two (medium), and four (deep). Three numbers of RNN units in each layer were tested: 128 (small), 256 (medium), and 512 (large). The model with medium model depth and large layer size achieved the best performance, in terms of both natural-language requirement comprehensibility and semantic linking, indicating that the model with medium model depth and large layer size is most suitable for the scale of the training data used – 7,500 sentences. For a training dataset significantly different in scale than this dataset, other model sizes could be tested and used.

The model training process was optimized. For example, three recurrent dropout rates were tested: 0, 0.2, and 0.4. The model achieved the highest performance when the recurrent dropout rate was set as 0.2, in terms of both requirement sentence comprehensibility and semantic linking, which indicates that models with no or very little dropout might overfit to the training data used whereas models with large dropout might underfit to the data.

7.3 Ablation analysis

7.3.1 Impact of semantic representation

To evaluate the proposed representation (i.e., the MFS requirement hierarchy), we tested an additional representation for the input, an SIE-based template representation, to serve as a baseline for comparative evaluation Each of the templates consists of a series of slots corresponding to the SIEs. For example, a template for quantitative requirement units consists of six types of slots that are corresponding to the following SIEs: compliance checking attribute, subject, subject relation, comparative relation, quantity value, quantity unit. To form the input to the requirement unit-to-text generation model using a template, the slots in the template were filled with the input SIEs accordingly. Thus, compared to the input in the form of the requirement hierarchy, the input generated using the templates has the same semantic information (i.e., the same set of SIEs) but different representation structures (i.e., it is a flat structure without defining units and linking the units into the requirement hierarchy). The same optimized hyperparameters (as shown in Table 2) and training and testing data were used for this experiment.

As shown in Table 3, the proposed representation achieved better performance. In terms of natural-language requirement comprehensibility, it outperformed the baseline representation by 3.1% in $BLEU_1$, 3.7% in $BLEU_2$, 3.2% in $ROUGE_1$, and 2.1% in $ROUGE_l$. In terms of semantic linking, it outperformed the baseline by 3.4% in precision, 2.5% in recall, and 2.9% in F1 measure. The results indicate that the proposed representation better captured the hierarchically complex semantic and syntactic structures in the requirements, which helped improve the model's ability to generate the intelligent code.

Table 3. Impact of Proposed Semantic Representation on Intelligent Code Generation

Representation	Natural-language requirement comprehensibility				Semantic linking		
	$BLEU_1$	$BLEU_2$	$ROUGE_1$	$ROUGE_l$	Precision	Recall	F_1 measure
Proposed representation (MFS requirement hierarchy)	**94.2%**	**90.5%**	**95.6%**	**93.1%**	**91.9%**	**92.3%**	**92.1%**
Baseline representation (template representation)	91.1%	86.8%	92.4%	91.0%	88.5%	89.8%	89.2%

[1]Bolded font indicates the highest performance.

7.3.2 Impact of semantic correspondence measurement and copying mechanism

Two different intelligent code generation methods with their models were tested for comparative evaluation: using the proposed correspondence score with the copying mechanism in the model (the proposed method) and without using any of the correspondence score or the copying mechanism in the model but only the RNN-based encoder-decoder model [Eqs.(1)-(4)] (the baseline). The alignment weights [Eq.(2)] of the model were used instead for semantic linking: the token in the output sentence segment is linked to the input token that has the highest alignment weight [57]. The same optimized hyperparameters (as shown in Table 2) and training and testing data were used for this evaluation.

As shown in Table 4, the proposed method achieved better performance. In terms of natural-language requirement comprehensibility, it outperformed the baseline method by 17.0% in $BLEU_1$, 18.9% in $BLEU_2$, 15.1% in $ROUGE_1$, and 14.6% in $ROUGE_l$. In terms of semantic linking, it outperformed the baseline method by 13.3% in precision, 12.9% in recall, and 13.1% in F1 measure. The results indicate that the proposed semantic correspondence score and the adopted copying mechanism helped improve the model's ability to generate the intelligent code.

596 **Table 4.** Impact of Correspondence Score and Copying Mechanism on Intelligent Code Generation

Method	Natural-language requirement comprehensibility				Semantic linking		
	$BLEU_1$	$BLEU_2$	$ROUGE_1$	$ROUGE_l$	Precision	Recall	F_1 measure
Proposed method (using correspondence score and copying mechanism)	**94.2%**	**90.5%**	**95.6%**	**93.1%**	**91.9%**	**92.3%**	**92.1%**
Baseline method (not using correspondence score and copying mechanism)	77.2%	71.6%	80.5%	78.5%	78.6%	79.4%	79.0%

597 [1]Bolded font indicates the highest performance.

598 ### *7.4 Flexibility analysis*

599 #### *7.4.1 Performance across different types of codes and standards*

600 To test the flexibility of the proposed method, i.e., test its performance across different regulatory
601 documents, the proposed method was used to generate three different types of intelligent requirements:
602 using MFS requirement hierarchies developed based on IBC, IECC, and ADA Standards. This is
603 important to evaluate, because requirements in different codes/standards typically have different semantic
604 and syntactic structures (e.g., terminology, vocabulary, and sentence length). As shown in Table 5, the
605 proposed method achieved consistently high performance across all three types of codes/standards, in
606 terms of both natural-language requirement comprehensibility (i.e., over 80.0% for all BLEU and
607 ROUGE scores) and semantic linking (i.e., over 88% for precision, recall, and F1 measure), indicating
608 that the approach has a good level of flexibility in dealing with different types of codes/requirements.

609 **Table 5.** Performance of the Proposed Approach Across Different Types of Codes/Requirements

Code/standard	Natural-language requirement comprehensibility				Semantic linking		
	$BLEU_1$	$BLEU_2$	$ROUGE_1$	$ROUGE_l$	Precision	Recall	F_1 measure
International Building Code (IBC)	96.2%	91.0%	96.2%	94.4%	93.4%	95.2%	94.1%
International Energy Conservation Code (IECC)	86.1%	80.3%	88.5%	85.4%	88.9%	89.5%	89.2%
Americans with Disabilities Act Standards for Accessible Design (ADA Standards)	87.7%	80.8%	90.4%	84.0%	89.6%	90.1%	89.8%

610

7.4.2 Performance across different levels of computability

To further evaluate its performance with respect to requirement computability, the proposed method was used to generate intelligent requirements with three different levels of computability: moderately high, moderately low, and low. These are the top three types of sentences that appear most frequently in building codes in terms of computability (e.g., they account for 22%, 39%, and 23% of a corpus of sentences from IBC and its amendments, respectively) [53]. The lower the level of computability, the more complex the semantic and syntactic structures of the sentences.

As shown in Table 6, the proposed method achieved consistently high performance (i.e., over 85% for all BLEU and ROUGE scores, and over 88% for precision, recall, and F1 measure) across all three computability levels, in terms of both natural-language requirement comprehensibility and semantic linking, indicating that the method has a good level of flexibility in generating requirements of various levels of computability. Also, all three selected types of requirements have hierarchical complex semantic and syntactic structures [3,53], indicating that the method is able to deal with such structures effectively.

Table 6. Performance of the Proposed Approach Across Different Code Computability Levels

Computability of requirements	Natural-language requirement comprehensibility				Semantic linking		
	$BLEU_1$	$BLEU_2$	$ROUGE_1$	$ROUGE_l$	Precision	Recall	F_1 measure
Moderately high	92.2%	88.4%	95.0%	90.6%	92.3%	93.6%	93.2%
Moderately low	89.0%	85.5%	91.4%	87.3%	88.3%	89.2%	88.6%
Low	91.6%	85.4%	92.9%	89.4%	89.6%	90.1%	89.8%

7.5 Comparison to a rule-based method

The rule-based method for NLG based on structured data by Bauer et al. [69] was used to serve as a baseline method for comparative evaluation of the generated natural language requirements (no semantic linking is needed for the rule-based approach because the generated sentences are directly constructed using the input SIEs, hence the comparison of semantic linking is not relevant).

Only the sentences from the IBC were used when developing the rules, to make the rule-based method and the proposed method, which was trained on sentences from the IBC only, comparable. Example rules

include: (1) Sentence subject rule: "IF (S is NOT None) AND (A is NOT None), THEN (S_{sub} = T_A + "of" + T_S OR S_{sub} = T_S + "with" + T_A"); (2) Sentence quantity rule: "IF (C is NOT None) AND (QV is NOT None) & (QU is NOT None), THEN S_q = T_C + T_{QV} + T_{QU}"; (3) Sentence predicate rule: "IF (S_{sub} is NOT None) AND (SR is NOT None) AND (S_q is NOT None), THEN S = S_{sub} + T_{SR} + S_q", where S_{sub}, S_q, S, are the partial sentences built using the sentence subject rule, sentence quantity rule, and sentence predicate rule, respectively; T_A, T_S, T_C, T_{QV}, T_{QU}, T_{SR} are the textual input corresponding to compliance checking attribute (A), subject (S), comparative relation (C), quantity value (QV), quantity unit (QU), and subject relation (SR), respectively.

As shown in Table 7, the proposed method achieved better performance. In terms of natural-language requirement comprehensibility, it outperformed the baseline method by 29.0% in $BLEU_2$, 7.4% in $ROUGE_1$, and 36.9% in $ROUGE_l$. The baseline's drop in $ROUGE_l$ is because compared to the gold standard sentences, although the sentences generated by the baseline method consist of a similar set of words, they are organized in a different way that is semantically less meaningful/correct, especially when the sentences have complex syntactic or semantic structures, as shown in the following example. Compared to the baseline method, the proposed method generated sentences that (1) capture more words that are seen in the gold standard sentences (indicated by the higher $ROUGE_1$), and (2) share similar semantic and syntactic structures with gold standard sentences (indicated by the higher $BLEU_2$ and $ROUGE_l$), as shown in the examples in Table 8.

Table 7. Impact of Correspondence Score and Copy Mechanism on Intelligent Code Generation

Method	Natural-language requirement comprehensibility			
	$BLEU_1$	$BLEU_2$	$ROUGE_1$	$ROUGE_l$
Proposed method	94.2%	**90.5%**	**95.6%**	**93.1%**
Baseline method	**99.3%**	61.5%	88.2%	56.2%

[1]Bolded font indicates the highest performance.

655 **Table 8.** Example Gold Standard and Generated Requirement Sentences

Example	Requirement source	Requirement sentence
Example 1	Generated by the baseline method	"maximum 19 mm height of existing or altered thresholds with slope not steeper than 0.5 have beveled edge on each side shall not be required with 404.2.5"
	Generated by the proposed method	"existing or altered thresholds 19 mm high maximum that have a beveled edge on each side with not steeper than 1:2 slope shall not be required to comply with the 404.2.5"
	Gold standard	"existing or altered thresholds 19 mm high maximum that have a beveled edge on each side with a slope not steeper than 1:2 shall not be required to comply with 404.2.5" [70]
Example 2	Generated by the baseline method	"with area of not more than 22500 square feet from any point in smoke compartment to smoke barrier door and travel distance shall not exceed 200 feet such stories shall be divided into smoke compartments"
	Generated by the proposed method	"such stories shall be divided into the smoke compartments with an area of not more than 22500 square feet and travel distance from any point in the smoke compartment to a smoke barrier door shall not exceed 200 feet"
	Gold standard	"such stories shall be divided into smoke compartments with an area of not more than 22500 square feet and the travel distance from any point in a smoke compartment to a smoke barrier door shall not exceed 200 feet" [52]

656 ### *7.6 Error analysis*

657 Four main sources of sentence generation errors were identified based on the analysis of the experimental
658 results: training and testing corpus noises, training data annotation errors, out-of-vocabulary tokens, and
659 structural complexity. First, the training and testing corpus were developed using regulatory documents
660 crawled from webpages and converted from PDF files, resulting in addition of noise during the data
661 crawling and conversion processes. For example, building codes typically contain a significant amount of
662 non-textual data such as tables and equations, some of which are difficult to be separated from the
663 requirement sentences and thus remain in the text files as noise. Second, the training data that were used
664 to train and evaluate the requirement unit-to-text generation model have errors, because they were
665 automatically annotated by pretrained regulatory information extraction models, which have not achieved
666 perfect (100%) performance. For example, the pretrained models tend to have errors in dealing with
667 multiword expressions (e.g., "path of egress"), each of which shall be annotated as a single SIE but could
668 be mistakenly annotated as two (e.g., "of" in the subject "path of egress" is not corrected linked). Third,
669 the requirement unit-to-text generation model in the proposed method could generate flawed requirement

sentence segments when the input requirement hierarchy contains out-of-vocabulary words (i.e., words not contained in the training data). For example, during testing, the method performed worse on requirements from IECC than those from ADA Standards and IBC because IECC has a relatively different vocabulary than those of ADA Standards and IBC. For instance, the proposed method failed when dealing with the compliance checking attribute ("area-weighted average maximum fenestration U-factor"), the quantitative values and units, and the reference ("tradeoffs from Section R402.1.5 or R405") in the gold standard "the area-weighted average maximum fenestration U-factor permitted using tradeoffs from Section R402.1.5 or R405 shall be 0.48 in Climate Zones 4 and 5 and 0.40 in Climate Zones 6 through 8 for vertical fenestration, and 0.75 in Climate Zones 4 through 8 for skylights" [71]. Fourth, the requirement-to-text generation model in the method could generate flawed requirement sentence segments when the target requirements have very high structural complexity. For example, the model might fail when dealing with sentence characteristics that indicate high syntactic complexity (e.g., complex noun phrases, verb phrases, and preposition phrases, and clauses of different types) or high semantic complexity (e.g., having multiple references and restrictions). For instance, the proposed method failed to capture the multiple subjects (e.g., "treatment rooms") and the compliance checking attribute ("aggregate area") in the gold standard "the aggregate area of corridors, patient rooms, treatment rooms, lounge or dining areas and other low-hazard areas on each side of each smoke barrier" [52], which is represented within a very complex noun and preposition phrase.

8 Limitations

Two limitations of the research are acknowledged. First, although the proposed method has been well supported by the proposed requirement hierarchy, further research is needed to study the usability of the proposed NLG-based approach when used in generating new requirements (i.e., new requirements, which are not in existing codes). In future work, the authors plan to develop a user-friendly BIM-integrated graphical user interface for capturing user input (i.e., the semantic information for the requirement hierarchies) and to study the usability of the proposed approach using such an interface. When

transforming existing codes to intelligent codes, no user input is needed, and the requirement hierarchies could be automatically generated using existing algorithms [31,54]. Second, although the proposed representation and method showed successful performance on requirements with different levels of computability and from different building codes and standards, the testing and evaluation did not cover all possible types of requirements, especially the challenging ones such as requirements that have hidden dependencies or assumptions, requirements that have ambiguities and require human judgment by nature, and requirements that have very complex syntactically and semantically structures. Further research is, thus, needed to (1) study the limit of the machine learning-based approach in the intelligent code generation, (2) better understand the benefits and challenges of machine learning-based methods compared to rule-based methods, especially when dealing with these challenging types of requirements, and (3) further refine, improve, and adapt the proposed representation and method based on these findings.

9 Contribution to the body of knowledge

This research is important from both intellectual and application perspectives. From an intellectual perspective, this research contributes to the body of knowledge in three primary ways. First, this research proposes a novel NLG-based approach for intelligent building code representation. It models intelligent requirements as natural-language requirements connected with their corresponding multi-form semantic (MFS) requirement hierarchies. The MFS requirement hierarchy is a new semantic representation of requirements for representing, analyzing, and generating requirements, especially the hierarchically complex ones. Its two forms, the forward and backward forms, support automated generation of intelligent code by facilitating the capturing of the semantic and syntactic structures of the requirements and the automated conversion and linking of the structured semantic information into natural-language sentences. The proposed intelligent code can circumvent the error-prone information extraction and transformation processes in the ACC systems and bring together the comprehensibility of the natural language with the computer-processability of the semantic representations. Second, this research is the first effort to automatically generate intelligent code given the regulatory information that defines the

720 requirements. It proposes a deep learning and NLG-based method, where an RNN-based sequence-to-
721 sequence model is adopted to generate requirement sentence segments and a semantic correspondence
722 score with a depth-first insertion and concatenation algorithm is defined to connect the segments into
723 whole requirements. The experimental results show that the proposed method achieved consistent
724 performance across intelligent requirements from different codes/standards and with different levels of
725 computability (i.e., from high to low computability), in terms of both natural language requirement
726 comprehensibility and semantic linking correctness. Third, it offers an approach for generating intelligent
727 code, with minimal development effort. The proposed method achieves competitive performance
728 compared to semantic and rule-based methods, while eliminating the need for handcrafting rules for
729 generating intelligent code. It further minimizes the manual effort for creating annotated data in training
730 the deep learning-based NLG models by leveraging pretrained domain-specific information extraction
731 and semantic relation extraction models and rules to automatically create large-scale annotated data to
732 train the requirement unit-to-text model.

733 From a practical perspective, first, the proposed intelligent code could help reduce ACC errors, improve
734 requirement comprehensibility, and facilitate intelligent analytics of building codes. All would lead to
735 enhanced project efficiency (e.g., by reducing time and cost of ACC) and fewer violations of building
736 codes and standards. Second, the application of the proposed NLG-based approach could be extended to
737 support many other applications and purposes such as automated generation of contract documents like
738 specifications and agreements, project planning documents like site plans, and progress reports. We can
739 envision many more applications of NLG in the AEC domain if combined with other artificial
740 intelligence (AI) approaches, like computer vision (e.g., automated generation of progress reports based
741 on site images).

742 **10 Conclusions and future work**

743 In this paper, a deep learning-based approach for generating intelligent building codes was proposed. First,
744 a new semantic representation of requirements, multi-form semantic (MFS) requirement hierarchy, was

745 proposed to support seamless and automated natural-language requirement generation. An MFS
746 requirement hierarchy represents a requirement in a hierarchical structure that consists of requirement
747 units and the relations between these units. Each requirement unit is further defined by several SIEs, such
748 as subject, compliance checking attribute, quantity value, and quantity unit. The requirement hierarchy is
749 represented in two supplementary forms: the surface form, which shows the units, relations, and SIEs and
750 thus can be used for requirement editing and development purposes, and the background form, which
751 shows the predicate-argument structures of the SIEs in a sequential format that can be directly fed into the
752 deep learning unit-to-text model for requirement generation purpose. Second, an intelligent code was
753 defined as a set of natural-language requirements connected with their corresponding requirement
754 hierarchies. An intelligent requirement consists of three parts – the natural-language requirement, its
755 corresponding requirement hierarchy, and the semantic correspondence keys that indicate the
756 correspondence between the requirement hierarchy and the natural-language requirement. Third, a deep
757 learning and semantic NLG-based method for generating intelligent building-code requirements was
758 proposed, which consists of three primary steps, requirement sentence segments generation, semantic
759 linking, and requirement configuration.

760 The requirement unit-to-text generation model was trained on training data automatically annotated using
761 pretrained information extraction models, which consist of 7,500 sentences, and tested on testing data
762 manually created that consisted of 600 sentences. Requirement sentence segments were generated using
763 the trained model, linked to the MFS requirement hierarchies based on semantic correspondence scores
764 and keys, and then configured into whole intelligent requirements. The comprehensibility of the generated
765 natural-language requirements was then evaluated using BLEU and ROUGE metrics, and the semantic
766 linking correctness of the links was evaluated using precision, recall, and F1 measure. A $BLEU_1$ of 94.2%,
767 $BLEU_2$ of 90.5%, $ROUGE_1$ of 95.6%, and $ROUGE_l$ of 93.1%, and a precision of 91.9%, recall of 92.3%,
768 and F1 measure of 92.1% were achieved, with the optimized hyperparameters. The ablation analysis
769 results indicate that the proposed requirement hierarchy and the proposed semantic correspondence

770 measurement, along with the adopted copying mechanism, are effective in generating intelligent code.
771 The flexibility analysis results indicate that the proposed method performed consistently on requirements
772 from different types of codes/standards and with different levels of computability from high (with
773 relatively simple semantic and syntactic structures) to low (with relatively complex semantic and
774 syntactic structures).

775 In future work, the authors plan to focus on improving the proposed intelligent code and the generation
776 method in three directions. First, the authors will explore the alignment of the proposed semantic
777 representation (i.e., the MFS requirement hierarchy) with the IFC schema. This would require the
778 matching and alignment of the regulatory concepts and the BIM/IFC concepts, using a machine learning-
779 based, rule-based, or hybrid approach, along with an ontology to support the semantic similarity analysis
780 and matching. Such alignment efforts could also be further incorporated within the intelligent code
781 generation process to have the resulting intelligent code readily aligned with the BIM. This would help
782 add an additional layer of intelligence for the code, which would not only be both human-comprehensible
783 and computer-understandable but also intelligently aligned with the BIM. Second, the deep learning-
784 based requirement sentence segment generation model could be improved by exploring different types of
785 model structures, such as transformer-based models (e.g., Bidirectional Encoder Representations from
786 Transformers) and model hyperparameters (e.g., activation functions such as ReLU and GeLU), and
787 incorporating more diversified syntactic and semantic patterns in the training data (e.g., including
788 different types of regulatory documents). Third, and most importantly, the authors will conduct additional
789 studies to further evaluate the practicality (e.g., in terms of time and cost) of using the proposed approach
790 in transforming natural-language building codes into intelligent codes, and combine the proposed
791 intelligent code with downstream ACC processes (e.g., BIM-regulatory information alignment and
792 semantic representation-based compliance reasoning) and existing semantic representations of
793 requirements (e.g., logic) in an integrated ACC system. Our ultimate goal is to leverage NLG, deep
794 learning, and other artificial intelligence approaches to reach a level where we can automatically and

effectively generate and use intelligent building codes for supporting fully automated compliance checking and other intelligent analytics processes in the AEC domain.

11 Data Availability Statement

The labeled gold standard data generated and used during the study are available from this link: https://publish.illinois.edu/rzhang65-data-sharing/.

12 Acknowledgements

The authors would like to thank the National Science Foundation (NSF). This material is based on work supported by the NSF under Grant No. 1827733. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

13 References

[1] Solibri. (2020). "Solibri Model Checker." https://www.solibri.com/products/solibri-model-checker. (Dec 15, 2020)

[2] Zhang, J., and El-Gohary, N. 2017. Semantic-based logic representation and reasoning for automated regulatory compliance checking. Journal of Computing in Civil Engineering, 31(1), 10.1061/(ASCE)CP.1943-5487.0000583.

[3] Zhou, P., and El-Gohary, N. 2017. "Ontology-based automated information extraction from building energy conservation codes." Automation in Construction, 74, 103-117.

[4] Nawari, N.O. 2019. "A Generalized Adaptive Framework (GAF) for Automating Code Compliance Checking." Buildings, 9(4), 86.

[5] Zhang, J., and El-Gohary, N. 2013. "Semantic NLP-based information extraction from construction regulatory documents for automated compliance checking." Journal of Computing in Civil Engineering, 10.1061/(ASCE)CP.1943-5487.0000346, 04015014.

[6] Solihin, W. and Eastman, C.M. 2016. "A knowledge representation approach in BIM rule requirement analysis using the conceptual graph." ITcon, 21, 370-401.

[7] Preidel, C., and Borrmann, A. 2016. "Towards code compliance checking on the basis of a visual programming language." ITcon. 21(25), 402-421.

[8] Wang, N. and Issa, R.R. 2020. "Natural Language Generation from Building Information Models for Intelligent NLP-based Information Extraction." In 27th EG-ICE International Workshop on Intelligent Computing in Engineering 2020. Universitatsverlag der TU Berlin. 275-284.

[9] Berlanga, R., Nebot, V. and Pérez, M. 2015. "Tailored semantic annotation for semantic search." Journal of Web Semantics, 30, 69-81.

[10] Gao, G., Liu, Y.S., Lin, P., Wang, M., Gu, M. and Yong, J.H. 2017. "BIMTag: Concept-based automatic semantic annotation of online BIM product resources." Advanced Engineering Informatics, 31, 48-61.

[11] Wang, Q., Pan, X., Huang, L., Zhang, B., Jiang, Z., Ji, H. and Knight, K. 2018. "Describing a knowledge base." arXiv preprint arXiv:1809.01797.

[12] Novikova, J., Dušek, O. and Rieser, V., 2017. The E2E dataset: New challenges for end-to-end generation. arXiv preprint arXiv:1706.09254.

[13] Wiseman, S., Shieber, S.M. and Rush, A.M., 2017. Challenges in data-to-document generation. arXiv preprint arXiv:1707.08052.

[14] Wang, H., 2020. Revisiting challenges in data-to-text generation with fact grounding. arXiv preprint arXiv:2001.03830.

[15] Garrett Jr, J.H. and Hakim, M.M. 1992. "Object-oriented model of engineering design standards." J. Comput. Civil. Eng., 6(3), 323-347.

[16] Ozkaya, I., and Akin, Ö. 2006. "Requirement-driven design: assistance for information traceability in design computing." Design Studies, 27(3), 381-398.

[17] Yurchyshyna, A., and Zarli, A. 2009. "An ontology-based approach for formalisation and semantic organisation of conformance requirements in construction." Autom. Construct., 18(8), 1084-1098.

[18] Pauwels, P., Van Deursen, D., Verstraeten, R., De Roo, J., De Meyer, R., Van de Walle, R. and Van Campenhout, J., 2011. A semantic rule checking environment for building performance checking. Automation in construction, 20(5), pp.506-518.

[19] Lee, J.K., Eastman, C.M. and Lee, Y.C. 2015. "Implementation of a BIM domain-specific language for the building environment rule and analysis." Journal of Intelligent & Robotic Systems, 79(3-4), 507-522.

[20] Uhm, M., Lee, G., Park, Y., Kim, S., Jung, J. and Lee, J.K., 2015. Requirements for computational rule checking of requests for proposals (RFPs) for building designs in South Korea. Advanced Engineering Informatics, 29(3), pp.602-615.

[21] Dimyadi, J., Pauwels, P. and Amor, R., 2016. Modelling and accessing regulatory knowledge for computer-assisted compliance audit. Journal of Information Technology in Construction, 21, pp.317-336.

[22] Weise, M., Liebich, T., Nisbet, N. and Benghi, C. 2017. "IFC model checking based on mvdXML 1.1." eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2016, 19-26.

[23] Hjelseth, E., and Nisbet, N. 2010. "Exploring semantic based model checking." http://itc.scix.net/data/works/att/w78-2010-54.pdf (Dec 15, 2020).

[24] LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. Nature. 521(7553), pp. 436. https://doi.org/10.1038/nature14539.

[25] Zhou, P., Qi, Z., Zheng, S., Xu, J., Bao, H. and Xu, B., 2016. Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling. arXiv preprint arXiv:1611.06639.

[26] Huang, Z., Xu, W. and Yu, K., 2015. Bidirectional LSTM-CRF models for sequence tagging. arXiv preprint arXiv:1508.01991.

[27] Dong, L. and Lapata, M., 2018. Coarse-to-fine decoding for neural semantic parsing. arXiv preprint arXiv:1805.04793.

[28] Bahdanau, D., Cho, K. and Bengio, Y., 2014. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

[29] Pan, Y. and Zhang, L., 2020. BIM log mining: Learning and predicting design commands. Automation in Construction, 112, p.103107.

[30] Zhong, B., Xing, X., Luo, H., Zhou, Q., Li, H., Rose, T. and Fang, W. 2020. "Deep learning-based extraction of construction procedural constraints from construction regulations." Advanced Engineering Informatics, 43, p.101003.

[31] Zhang, R. and El-Gohary, N., 2021. A deep neural network-based method for deep information extraction using transfer learning strategies to support automated compliance checking. Automation in Construction, 132, p.103834.

[32] Gatt, A. and Krahmer, E. 2018. "Survey of the state of the art in natural language generation: Core tasks, applications and evaluation." Journal of Artificial Intelligence Research, 61, 65-170.

[33] Wen, T.H., Gasic, M., Mrksic, N., Su, P.H., Vandyke, D. and Young, S. (2015). "Semantically conditioned lstm-based natural language generation for spoken dialogue systems." arXiv preprint arXiv:1508.01745.

[34] You, Q., Jin, H., Wang, Z., Fang, C. and Luo, J. 2016. "Image captioning with semantic attention." Proc. CVPR IEEE. 4651–4659.

[35] Zhang, H., Xu, J. and Wang, J. 2019. "Pretraining-based natural language generation for text summarization." arXiv preprint arXiv:1902.09243.

[36] Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S. and Zhang, Z. (2018). "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task." arXiv preprint arXiv:1809.08887.

[37] Puduppully, R., Dong, L. and Lapata, M. 2019. "Data-to-text generation with content selection and planning." In Proceedings of the AAAI Conference on Artificial Intelligence. 33, 6908-6915.

[38] Gu, J., Lu, Z., Li, H. and Li, V.O. 2016. "Incorporating copying mechanism in sequence-to-sequence learning." arXiv preprint arXiv:1603.06393.

[39] Nie, F., Wang, J., Yao, J.G., Pan, R. and Lin, C.Y. 2018. "Operations guided neural networks for high fidelity data-to-text generation." arXiv preprint arXiv:1809.02735.

[40] See, A., Liu, P.J. and Manning, C.D. 2017. "Get to the point: Summarization with pointer-generator networks." arXiv preprint arXiv:1704.04368.

[41] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I. 2017. "Attention is all you need." In Advances in neural information processing systems. 5998-6008.

[42] Mager, M., Astudillo, R.F., Naseem, T., Sultan, M.A., Lee, Y.S., Florian, R. and Roukos, S. 2020. "GPT-too: A language-model-first approach for AMR-to-text generation." arXiv preprint arXiv:2005.09123.

[43] Ye, R., Shi, W., Zhou, H., Wei, Z. and Li, L. 2020. "Variational Template Machine for Data-to-Text Generation." arXiv preprint arXiv:2002.01127.

[44] Thewalt, C. and Moskowitz, D. 1990. "Automated text generation for building standards." J. Comput. Civ. Eng., 4(1), 20-36.

[45] Ryoo, B.Y., Skibniewski, M.J. and Kwak, Y.H. 2010. "Web-based construction project specification system." J. Comput. Civ. Eng. 24(2), 212–221.

[46] Avitru. (2020). Spec Editor, https://avitru.com/software/spec-editor. (Dec 15, 2020)

[47] Digicon. 2020. BIMdrive Specification Management Software, http://www.digicon.ab.ca/services.aspx. (Dec 15, 2020).

[48] ISO. 2021. "ISO/DIS 29481-3(en) Building information models — Information delivery manual — Part 3: Data schema and code." https://www.iso.org/obp/ui/#iso:std:iso:29481:-3:dis:ed-1:v1:en.

[49] Ferreira, T.C., van der Lee, C., Van Miltenburg, E. and Krahmer, E., 2019. Neural data-to-text generation: A comparison between pipeline and end-to-end architectures. arXiv preprint arXiv:1908.09022.

[50] Schmitt, M., Sharifzadeh, S., Tresp, V. and Schütze, H., 2019. An unsupervised joint system for text generation from knowledge graphs and semantic parsing. arXiv preprint arXiv:1904.09447.

[51] Sutskever, I., Vinyals, O. and Le, Q.V. 2014. "Sequence to sequence learning with neural networks." Adv. Neur. In. 3104-3112.

[52] ICC (International Code Council). 2018. 2018 International Building Code. ICC. Washington, D.C.

[53] Zhang, R., and El-Gohary, N., 2020. Clustering-based Approach for Building Code Computability Analysis. Journal of Computing in Civil Engineering. https://doi.org/10.1061/(ASCE)CP.1943-5487.0000967.

[54] Zhang, R., and El-Gohary, N., 2021. Hierarchical representation and deep learning-based method for automatically transforming textual building codes into semantic computable requirements. Journal of Computing in Civil Engineering.

[55] Clark, V., and Creswell, J. 2008. The mixed methods readers, Sage Publications, Thousand Oaks, CA.

[56] Pestian, J.P., Deleger, L., Savova, G.K., Dexheimer, J.W., Solti, I. 2012. Natural language processing—the basics. Pediatric Biomedical Informatics: Computer Applications in Pediatric Research, Springer, Netherlands, Dordrecht, pp. 149-172

[57] Luong, M.T., Pham, H. and Manning, C.D., 2015. Effective approaches to attention-based neural machine translation. arXiv preprint arXiv:1508.04025.

[58] Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R. and Schmidhuber, J. 2016. "LSTM: A search space odyssey." IEEE transactions on neural networks and learning systems, 28(10), 2222-2232.

[59] Graves, A., Mohamed, A.R. and Hinton, G. 2013. Speech recognition with deep recurrent neural networks. In 2013 IEEE international conference on acoustics, speech and signal processing (pp. 6645-6649). IEEE.

[60] Jurafsky, D. and Martin, J.H. 2014. Speech and language processing (Vol. 3). US: Prentice Hall.

[61] Wiseman, S. and Rush, A.M., 2016. Sequence-to-sequence learning as beam-search optimization. arXiv preprint arXiv:1606.02960.

[62] Le, Q.V., Jaitly, N. and Hinton, G.E., 2015. A simple way to initialize recurrent networks of rectified linear units. arXiv preprint arXiv:1504.00941.

[63] Papineni, K., Roukos, S., Ward, T. and Zhu, W.J. 2002. BLEU: a method for automatic evaluation of machine translation. Proc. 40th Annual Meeting on ACL. pp. 311–318.

[64] Madotto, A., Wu, C.S. and Fung, P., 2018. Mem2seq: Effectively incorporating knowledge bases into end-to-end task-oriented dialog systems. arXiv preprint arXiv:1804.08217.

[65] Wu, C.S., Socher, R. and Xiong, C., 2019. Global-to-local memory pointer networks for task-oriented dialogue. arXiv preprint arXiv:1901.04713.

[66] Dušek, O. and Kasner, Z., 2020. Evaluating semantic accuracy of data-to-text generation with natural language inference. arXiv preprint arXiv:2011.10819.

[67] Lin, C.Y. 2004. Rouge: a package for automatic evaluation of summaries. Proc. of ACL-04 Workshop. 8, pp. 74–81.

[68] Chen, W., Su, Y., Yan, X. and Wang, W.Y., 2020. KGPT: Knowledge-Grounded Pre-Training for Data-to-Text Generation. arXiv preprint arXiv:2010.02307.

[69] Bauer, A., Hoedoro, N. and Schneider, A., 2015. Rule-based Approach to Text Generation in Natural Language-Automated Text Markup Language (ATML3). In Challenge+ DC@ RuleML.

[70] U.S. Department of Justice. 2010. 2010 ADA Standards for Accessible Design. https://www.ada.gov/2010ADAstandards_index.htm (Dec 15, 2020)

[71] ICC (International Code Council). 2018. 2018 International Energy Conservation Code. ICC. Washington, D.C.