How Can ChatGPT Help in Automated Building Code Compliance Checking?

Jiansong Zhang, Ph.D.[1]

[1]School of Construction Management Technology, Purdue University, United States of America
zhan3062@purdue.edu

Abstract

One main challenge in the full automation of building code compliance checking is in the extraction and transformation of building code requirements into computable representations. Semantic rule-based approach has been taken mainly due to its expected better performance than machine learning-based approach on this particular task. With the recent advancement in deep learning AI, particularly the launch of ChatGPT by OpenAI, there is a potential for this landscape to be shifted given the highly regarded capabilities of ChatGPT in processing (i.e., understanding and generating) natural language texts and computer codes. In this paper, the author preliminarily explored the use of ChatGPT in converting (i.e., extracting and transforming) building code requirements into computer codes, and compared it with the results from cutting-edge semantic rule-based approach. It was found that comparing to the semantic rule-based approach, the conversion results from ChatGPT still has limitations, but there is a great potential for it to help speed up the implementation and scale-up of automated building code compliance checking systems.

Keywords –

Automated building code compliance checking; AI; ChatGPT; SNACC; Natural language processing

1 Introduction

Automated building code compliance checking has been one promising application of artificial intelligence (AI) since its inception. Earlier efforts, however, mainly hard-coded building code requirements into computable representations. While effective in addressing various types of building code requirements, the large amount of efforts required in such hard-coding tasks coupled with the large number and different types/versions of building codes adopted at different authorities having jurisdictions (AHJs), in fact prohibited automated building code compliance checking systems from easy scaling up or even implementation in the field in the first place. In the last two decades or so, efforts have been made to automatically or semi-automatically convert building code requirements from natural language texts into computable representations. Such efforts are considered critical in overcoming this major barrier to full automation of code compliance checking systems. Semi-automated approach requires manual labeling (or marking up) of the building code requirements using predefined tag set such as Requirement, Applicability, Selection, and Exceptions (RASE) [1] which was used in the *SMARTcodes builder* software of International Code Council [2]. While the improvement was salient comparing to the hard-coding approach, such semi-automated approach still failed to lead to wide implementation or scaling up of automated building code compliance checking systems. When it comes to investigating full automation of information extraction from building codes and their transformation into computable representations, two main approaches have been taken: machine learning-based and rule-based. In spite of the high initial rule development efforts required in the rule-based approach, it has been shown to achieve better performance than machine learning-based approach (e.g., 96.9% precision vs. 93.1% precision; 94.4% recall vs. 92.9% recall) [3,4]. Now, with the highly promising launch of GPTChat of OpenAI which inundated the AI community and our society at large recently, can the landscape of automated conversion of building codes into computable representations be shifted? To answer this question, the author conducted an initial but systematic investigation of the capabilities of GPTChat on this task, and compared it with the cutting-edge rule-based approach.

2 Background

In the automated processing (i.e., extraction and transformation) of building code requirements, for both rule-based approach and machine learning approach, natural language processing (NLP) techniques are employed. This section provides some relevant backgrounds in NLP, information extraction, building code compliance checking, and GPTChat.

2.1 Natural language processing

Natural language processing (NLP) aims to enable computers to understand and process natural human languages in a human-like manner [5]. It is an important field of AI, with multiple NLP-based systems benchmarking milestones of AI development such as the ELIZA ChatterBot program developed by Joseph Weizenbaum at MIT in the 1960s [6] and the IBM Watson that outperformed human champions in the "Jeopardy!" game in 2011 [7].

In construction research, NLP has been used to analyze and process various types of construction documents to: automate the classification of project documents [8]; extract key elements of change orders [9]; identify poisonous clauses or extract concepts and relations from construction contracts [10,11]; review and analyze construction specifications [12-15]; extract precursors and outcomes from construction injury reports [16]; and retrieve similar construction risk cases for project risk management [17]. Nora El-Gohary's group pioneered the use of NLP in analyzing and processing building codes [3,18-20].

2.1.1 Information Extraction

Information extraction is a classic task/application in NLP, together with others such as part-of-speech tagging, named entity recognition, word sense disambiguation, information retrieval, etc. While information retrieval has achieved huge success in the commercial sector as represented by popular search engines such as Google Search and Microsoft Bing Search, information extraction is a similar but slightly different task in terms of: (1) results being monotonous; and (2) task provided with predefined information template. Information extraction is typically the first step of converting building code requirements to computable representations, with machine learning and rule-based as the two main approaches used [3,4]. In addition, semantic modeling has played important roles in this task, by enabling/augmenting extraction algorithms with semantic relations and associations, and simplifying disambiguation at the word level and phrase level.

2.1.2 Machine Translation

Machine translation aims to translate one language into another using computers. It is an even more important task to help with automated conversion of building codes to computable representations, comparing to information extraction. Modern machine translation algorithms predominately took a machine learning approach. Because features are the main ingredient of machine learning models, and feature engineering can be a labor-intensive task, information extraction is sometimes used to help generate features for training machine learning models.

In spite of the maturity of machine translation tools and techniques, the direct application of machine learning models to conquering the building code conversion problem has been under-investigated.

2.2 Building code compliance checking

Building code compliance checking is traditionally an intelligent manual task that requires deep expertise in the architecture, engineering, and construction (AEC) domain. The manual compliance checking operation is time-consuming, costly, and subjective/error-prone [3,4]. Efforts in automating the code compliance checking task date back to the 1960s when Fenves et al. [21] encoded American Institute of Steel Construction (AISC) specifications into decision tables. Since then there have been many efforts in automating the compliance checking for various building regulations such as those summarized in [22,23]. However, these efforts/systems still hard-code building code requirements or at most provided pre-defined templates to allow some flexibility in defining rules to reflect code requirements. A more efficient and flexible conversion of building codes into computable representations would significantly increase the usability of such systems. In view of that, the author jumped on a journey to harness the power of NLP and other modeling and AI techniques to pursue full automation of building code compliance checking 12 years ago and created semantic NLP-based information extraction and transformation algorithms that can automatically convert building code requirements into logic programs.

2.3 GPTChat

"GPT-Chat is a GPT-3 based conversational AI that allows users to interact with the language model to generate human-like text. GPT-Chat uses OpenAI's GPT-3 model, which is a state-of-the-art language model that has been trained on a massive dataset of text data. It can generate text that is highly coherent and contextually appropriate, making it well-suited for a wide range of natural language processing tasks such as text generation, language translation, and question-answering." [24]. This is the definition provided by GPTChat when asked "What is GPTChat?"

Trained using Reinforcement Learning from Human Feedback (RLHF) and fine-tuned using Proximal Policy Optimization (PPO), GPTChat has been demonstrated strong capabilities (close to completely correct and only required minor manual adjustments) in generating and translating computer codes (especially Python and Java, at the time of test) based on natural language human instructions [25,26,27]. In the context of building code conversion, the following questions naturally emerge: How well can GPTChat convert building code requirements to computer codes? How will the performance compare to the state-of-the-art rule-based conversion?

3 Approach

This study is explorative in nature. A simple comparative evaluation approach is taken by comparing the conversion results on representative regulatory requirements.

4 Experiment

Regulatory requirements from International Building Codes in different types and in an increasing level of complexity were selected, consistent with the test cases developed in the author's previous studies for automated code compliance checking [3,18].

The tool used to generate logic programs is the most recent version of the semantic NLP-based automated compliance checking (SNACC) system which was initially described in [28,29] and has been constantly refined with more training data [30]. Although SNACC could batch process regulatory requirements, for better clarity in the comparative evaluation, the selected regulatory requirements were processed/tested one at a time. Then the same regulatory requirement was formed as a question/command in form of "*Please convert "…" into Python code.*" where the *"…"* will be replaced by the regulatory requirement (RQ) text with the double quotes surrounding it reserved (see an example in Fig. 1).

Time measurement was performed for both conversion processes on each RQ. When using the SNACC, time benchmarking function in Java was used to tell/record the start and finish times of the conversion process, whereas the processing time in GPTChat was measured using an online timer. Note that the GPTChat also automatically generated code explanations the time consumption of which was not included.

Convert "Occupiable spaces, habitable spaces and corridors shall have a ceiling height of not less than 7 feet 6 inches" into Python code.

Figure 1. Example question/command in GPTChat.

5 Experimental Results and Analysis

The experimental results are listed below.

RQ1: "Spacing of transverse reinforcement shall not exceed 8 inches." [31]

Logic rules generated by SNACC: *compliance_Spacing_of_transverse_reinforcement1(Transverse_reinforcement):-spacing(Spacing),transverse_reinforcement(Transverse_reinforcement),has(Transverse_reinforcement,Spacing),not greater_than(Spacing,quantity(8,inches)).*

check_Spacing_of_transverse_reinforcement1(X):-(compliance_Spacing_of_transverse_reinforcement1(X)->writeln((X,is,compliant,with,section,1904-3,rule1));writeln((X,is,noncompliant,with,section,1904-3,spacing,should,be,not,greater_than,8,inches,rule2))).

Python code generated by GPTChat: Figure 2.
Time consumption by SNACC: 21.32 s.
Time consumption by GPTChat: 12.75 s.

```python
max_transverse_reinforcement_spacing = 8 # in inches

def check_transverse_reinforcement_spacing(reinforcement):
    if reinforcement.spacing > max_transverse_reinforcement_spacing:
        print("Spacing of transverse reinforcement exceeds the maximum limit of 8 inches.")
    else:
        print("Spacing of transverse reinforcement meets the maximum limit of 8 inches.")
```

Figure 2. Python code for RQ1 by ChatGPT

Code explanation generated by GPTChat: "This code defines a variable **'max_transverse_reinforcement_spacing'** that holds the maximum allowed spacing of transverse reinforcement in inches, and a function **'check_transverse_reinforcement_spacing'** that takes an object as an input (could be a class or instance) that represent the transverse reinforcement and compares its spacing property to the maximum allowed spacing. If the spacing is greater than the maximum allowed spacing, the function will output a message indicating that the spacing exceeds the limit, otherwise it will output a message indicating that the spacing meets the limit."

RQ2: "Courts shall not be less than 3 feet in width." [32]

Logic rules generated by SNACC:
compliance_width_of_courts1(Courts):-courts(Courts),width(Width),has(Courts,Width),not less_than(Width,quantity(3,feet)).

check_width_of_courts1(X):-(compliance_width_of_courts1(X)->writeln((X,is,compliant,with,section,1904-3,rule1));writeln((X,is,noncompliant,with,section,1904-3,width,should,be,not,less_than,3,feet,rule2))).

Python code generated by GPTChat: Figure 3.
Time consumption by SNACC: 20.58 s.
Time consumption by GPTChat: 11.16 s.

```python
min_court_width = 3 # in feet

def check_court_width(court):
    if court.width < min_court_width:
        print("Court width does not meet the minimum requirement of 3 feet.")
    else:
        print("Court width meets the minimum requirement of 3 feet.")
```

Figure 3. Python code for RQ2 by ChatGPT

Code explanation generated by GPTChat: "This code defines a variable **'min_court_width'** that holds the minimum required court width in feet, and a function

‘check_court_width’ that takes a court as an input and compares its width to the minimum required width. If the width of the court is less than the minimum required width, the function will output a message indicating that the court width does not meet the requirement, otherwise it will output a message indicating that the court width meets the requirement.”

RQ3: “The unit shall have a living room of not less than 220 square feet of floor area.” [32]

Logic rules generated by SNACC:

compliance_unit1(Unit):-unit(Unit),living_room(Living_room),has(Unit,Living_room),floor_area(Floor_area),has(Living_room,Floor_area),not less_than(Floor_area,quantity(220,square_feet)).

check_unit1(X):-(compliance_unit1(X)->writeln((X,is,compliant,with,section,1904-3,rule1));writeln((X,is,noncompliant,with,section,1904-3,rule2))).

Python code generated by GPTChat: Figure 4.

Time consumption by SNACC: 20.45 s.

Time consumption by GPTChat: 12.41 s.

```python
min_living_room_area = 220  # square feet

def check_living_room_area(unit):
    if unit.living_room_area < min_living_room_area:
        print("Living room area does not meet the minimum requirement of 220 square feet.")
    else:
        print("Living room area meets the minimum requirement of 220 square feet.")
```

Figure 4. Python code for RQ3 by ChatGPT

Code explanation generated by GPTChat: “This code defines a variable **‘min_living_room_area’** that holds the minimum required living room area in square feet, and a function **‘check_living_room_area’** that takes a unit as an input and compares its living room area to the minimum required area. If the living room area of the unit is less than the minimum required area, the function will output a message indicating that the living room area does not meet the requirement, otherwise it will output a message indicating that the living room area meets the requirement.”

RQ4: “The thickness of concrete floor slabs supported directly on the ground shall not be less than 31/2 inches.” [31]

Logic rules generated by SNACC:

compliance_thickness_of_concrete_floor_slabs_supported1(Concrete_floor_slabs_supported):-thickness(Thickness),concrete_floor_slabs_supported(Concrete_floor_slabs_supported),has(Concrete_floor_slabs_supported,Thickness),ground_prep(Ground),on(Concrete_floor_slabs_supported,Ground),not less_than(Thickness,quantity(31/2,inches)).

check_thickness_of_concrete_floor_slabs_supported1(X):-(compliance_thickness_of_concrete_floor_slabs_supported1(X)->writeln((X,is,compliant,with,section,1904-3,rule1));writeln((X,is,noncompliant,with,section,1904-3,thickness,should,be,not,less_than,31/2,inches,rule2))).

Python code generated by GPTChat: Figure 5.

Time consumption by SNACC: 19.03 s.

Time consumption by GPTChat: 17.22 s.

```python
min_concrete_thickness = 3.5 # in inches

def check_concrete_thickness(slab):
    if slab.is_supported_directly_on_ground and slab.thickness < min_concrete_thickness:
        print("Thickness of concrete slab does not meet the minimum requirement of 3.5 inches for slabs supported directly on the ground.")
    else:
        print("Thickness of concrete slab meets the minimum requirement of 3.5 inches for slabs supported directly on the ground.")
```

Figure 5. Python code for RQ4 by ChatGPT

Code explanation generated by GPTChat: “This code defines a variable **‘min_concrete_thickness’** that holds the minimum required thickness of concrete floor slab supported directly on the ground in inches, and a function **‘check_concrete_thickness’** that takes an object as an input (could be a class or instance) that represent the slab, checks if it is supported directly on the ground and compares its thickness property to the minimum required thickness. If the thickness is less than the minimum required thickness, the function will output a message indicating that the slab does not meet the requirement, otherwise it will output a message indicating that the slab meets the requirement.”

RQ5: “Occupiable spaces, habitable spaces and corridors shall have a ceiling height of not less than 7 feet 6 inches.” [32]

Logic rule generated by SNACC:

compliance_Occupiable_spaces1(Occupiable_spaces):-(occupiable_spaces(Occupiable_spaces);habitable_spaces(Occupiable_spaces);corridors(Occupiable_spaces)),ceiling_height(Ceiling_height),has(Occupiable_spaces_habitable_spaces_corridors,Ceiling_height),not less_than(Ceiling_height,quantity(7,feet)),associated(Occupiable_spaces,Habitable_spaces),associated(Occupiable_spaces,Ceiling_height).

check_Occupiable_spaces1(X):-(compliance_Occupiable_spaces1(X)->writeln((X,is,compliant,with,section,1904-3,rule1));writeln((X,is,noncompliant,with,section,1904-

3,rule2))).

Python code generated by GPTChat: Figure 6.
Time consumption by SNACC: 18.04 s.
Time consumption by GPTChat: 17.99 s.

```
min_ceiling_height = 7 * 12 + 6  # convert 7 feet 6 inches to inches

def check_ceiling_height(space):
    if space.type in ["occupiable", "habitable", "corridor"] and
space.ceiling_height < min_ceiling_height:
        print("Ceiling height of space does not meet the minimum requirement of
7 feet 6 inches")
    else:
        print("Ceiling height of space meets the minimum requirement of 7 feet 6
inches")
```

Figure 6. Python code for RQ5 by ChatGPT

Code explanation generated by GPTChat: "This code defines a variable **'min_ceiling_height'** that holds the minimum required ceiling height in inches (7 feet 6 inches converted to inches), and a function **'check_ceiling_height'** that takes a space as an input, checks if it is of type "occupiable", "habitable", or "corridor", and compares its ceiling height to the minimum required height. If the ceiling height of the space is less than the minimum required height, the function will output a message indicating that the space does not meet the requirement, otherwise it will output a message indicating that the space meets the requirement."

RQ6: "Bars shall not be smaller than No. 4 and shall have a total area of not less than 0.002 times the gross cross-sectional area of the footing." [31]

Logic rule generated by SNACC:

compliance_Bars1(Bars):-bars(Bars),times_gross_cross_sectional_area(Times_gross_cross_sectional_area),footing(Footing),has(Footing,Times_gross_cross_sectional_area),not less_than_or_equal(Bars,quantity(1,no_4)).

compliance_Total_area3(Total_area):-total_area(Total_area),times_gross_cross_sectional_area(Times_gross_cross_sectional_area),footing(Footing),has(Footing,Times_gross_cross_sectional_area),not less_than(Total_area,quantity(0.002,Times_gross_cross_sectional_area)),associated(Total_area,Times_gross_cross_sectional_area).

check_Bars1(X):-(compliance_Bars1(X)->writeln((X,is,compliant,with,section,1904-3,rule1));writeln((X,is,noncompliant,with,section,1904-3,bars,should,be,not,less_than_or_equal,1,no_4,rule2))).

check_Total_area3(X):-(compliance_Total_area3(X)->writeln((X,is,compliant,with,section,1904-3,rule3));writeln((X,is,noncompliant,with,section,1904-3,total_area,should,be,not,less_than,0.002,times_gross_cross_sectional_area,rule4))).

Python code generated by GPTChat: Figure 7.
Time consumption by SNACC: 19.45 s.
Time consumption by GPTChat: 21.03 s.

```
min_bar_size = 4
min_bar_area_ratio = 0.002

def check_footing_reinforcement(footing):
    for bar in footing.bars:
        if bar.size < min_bar_size:
            print("Bar size does not meet the minimum requirement of
No. 4.")
            return
        if bar.area < footing.gross_cross_sectional_area *
min_bar_area_ratio:
            print("Total area of bars does not meet the minimum
requirement of 0.002 times the gross cross-sectional area of the
footing.")
            return
    print("Footing reinforcement meets the requirements.")
```

Figure 7. Python code for RQ6 by ChatGPT

Code explanation generated by GPTChat: "This code defines a variable **'min_bar_size'** that holds the minimum required size for bars in the footing, a variable **'min_bar_area_ratio'** that holds the minimum required ratio of total area of bars to the gross cross-sectional area of the footing, and a function **'check_footing_reinforcement'** that takes an object as an input (could be a class or instance) that represent the footing and check each bar in the footing against the minimum required size, and the minimum required ratio of total area of bars to the gross cross-sectional area of the footing. If any bar does not meet these requirements, the function will output a message indicating that the footing reinforcement does not meet the requirement, otherwise it will output a message indicating that the footing reinforcement meets the requirements."

RQ7: "Any room with a furred ceiling shall be required to have the minimum ceiling height in two thirds of the area thereof, but in no case shall the height of the furred ceiling be less than 7 feet." [32]

Logic rules generated by SNACC:

compliance_a_furred_ceiling1(A_furred_ceiling):-room(Room),a_furred_ceiling(A_furred_ceiling),has(Room,A_furred_ceiling),minimum_ceiling_height(Minimum_ceiling_height),area(Area),height(Height),furred_ceiling(Furred_ceiling),has(Furred_ceiling,Height),has(Furred_ceiling,A_furred_ceiling),have_in(Room,quantity(2/3,Area)),not less_than(Height,quantity(7,feet)).

check_a_furred_ceiling1(X):-(compliance_a_furred_ceiling1(X)->writeln((X,is,compliant,with,section,1904-3,rule1));writeln((X,is,noncompliant,with,section,1904-3,rule2))).

Python code generated by GPTChat: Figure 8.
Time consumption by SNACC: 20.54 s.

Time consumption by GPTChat: 24.45 s.

```
min_ceiling_height = 7 * 12 # convert 7 feet to inches
min_ceiling_height_ratio = 2/3

def check_furred_ceiling_height(room):
    if room.has_furred_ceiling:
        if room.ceiling_height < min_ceiling_height:
            print("Ceiling height of room does not meet the minimum
requirement of 7 feet")
        elif room.furred_ceiling_area < min_ceiling_height_ratio *
room.area:
            print("Furred ceiling area does not meet the minimum
requirement of two thirds of the room area")
        else:
            print("Ceiling height and furred ceiling area of room
meet the minimum requirements")
    else:
        print("Room does not have a furred ceiling")
```

Figure 8. Python code for RQ7 by ChatGPT

Code explanation generated by GPTChat: “This code defines a variable **‘min_ceiling_height’** that holds the minimum required ceiling height in inches (7 feet converted to inches), a variable **‘min_ceiling_height_ratio’** that holds the minimum required ratio of the area of furred ceiling to the room area, and a function ‘check_furred_ceiling_height’ that takes a room as an input, checks if it has a furred ceiling, if yes it compares its ceiling height and furred ceiling area to the minimum required”

RQ8: “In dwellings assigned to Seismic Design Category D or E, the height of the wall shall not exceed 8 feet, the thickness shall not be less than 71/2 inches, and the wall shall retain no more than 4 feet of unbalanced fill.” [31]

Logic rules generated by SNACC:

compliance_Height_of_wall1(Height):- dwellings(Dwellings),in_prep(Wall,Dwellings),assigned_to(Dwellings,Seismic_Design_Category_D_or_E),seismic_design_category_d_or_e(Seismic_Design_Category_D_or_E),height(Height),wall(Wall),has(Wall,Height),thickness(Thickness),unbalanced_fill(Unbalanced_fill),not greater_than(Height,quantity(8,feet)).

compliance_Thickness_of_wall3(Thickness):- dwellings(Dwellings),in_prep(Wall,Dwellings),assigned_to(Dwellings,Seismic_Design_Category_D_or_E),seismic_design_category_d_or_e(Seismic_Design_Category_D_or_E),height(Height),wall(Wall),thickness(Thickness),has(Wall,Thickness),unbalanced_fill(Unbalanced_fill),not less_than(Thickness,quantity(71/2,inches)).

compliance_Unbalanced_fill_of_wall5(Unbalanced_fill):- dwellings(Dwellings),in_prep(Wall,Dwellings),assigned_to(Dwellings,Seismic_Design_Category_D_or_E),seismic_design_category_d_or_e(Seismic_Design_Category_D_or_E),height(Height),wall(Wall),thickness(Thickness),unbalanced_fill(Unbalanced_fill),has(Wall,Unbalanced_fill),not greater_than(Unbalanced_fill,quantity(4,feet)).

check_Height_of_wall1(X):- (compliance_Height_of_wall1(X)->writeln((X,of,wall,is,compliant,with,section,1904-3,rule1));writeln((X,of,wall,is,noncompliant,with,section,1904-3,height,should,be,not,greater_than,8,feet,rule2))).

check_Thickness_of_wall3(X):- (compliance_Thickness_of_wall3(X)->writeln((X,of,wall,is,compliant,with,section,1904-3,rule3));writeln((X,of,wall,is,noncompliant,with,section,1904-3,thickness,should,be,not,less_than,71/2,inches,rule4))).

check_Unbalanced_fill_of_wall5(X):- (compliance_Unbalanced_fill_of_wall5(X)->writeln((X,of,wall,is,compliant,with,section,1904-3,rule5));writeln((X,of,wall,is,noncompliant,with,section,1904-3,unbalanced_fill,should,be,not,greater_than,4,feet,rule6))).

Python code generated by GPTChat: Figure 9.

Time consumption by SNACC: 20.50 s.

Time consumption by GPTChat: 27.52 s.

```
max_wall_height = 8 # in feet
min_wall_thickness = 7.5 # in inches
max_unbalanced_fill = 4 # in feet

def check_wall_requirement(dwelling):
    if dwelling.seismic_design_category in ["D", "E"]:
        if dwelling.wall_height > max_wall_height:
            print("Wall height exceeds the maximum limit of 8 feet for Seismic
Design Category D or E.")
        elif dwelling.wall_thickness < min_wall_thickness:
            print("Wall thickness is less than the minimum limit of 7.5 inches
for Seismic Design Category D or E.")
        elif dwelling.unbalanced_fill > max_unbalanced_fill:
            print("Wall retains more than 4 feet of unbalanced fill for Seismic
Design Category D or E.")
        else:
            print("Wall meets the requirements for Seismic Design Category D or
E.")
    else:
        print("This check is only applicable for Seismic Design Category D or
E.")
```

Figure 9. Python code for RQ8 by ChatGPT

Code explanation generated by GPTChat: “This code defines a variable **‘max_wall_height’** that holds the maximum allowed height of the wall in feet, a variable **‘min_wall_thickness’** that holds the minimum required thickness of the wall in inches, a variable **‘max_unbalanced_fill’** that holds the maximum allowed unbalanced fill that the wall should retain, and a function **‘check_wall_requirement’** that takes an object as an input (could be a class or instance) that represent the dwelling and checks if the dwelling is assigned to seismic design category”

5.1 Analysis

It was found that with the increase of complexity in the regulatory requirement, the processing time of SNACC stayed relatively flat whereas the processing time of GPTChat stably increased (Figure 10). This can be caused by the inherent complexity of GPTChat's deep learning models which is much more complex than the rule-based core of SNACC.

In terms of the quality of the conversion results, both SNACC and GPTChat captured the logic in each regulatory requirement reasonably well. One interesting observation is that the Python codes generated by GPTChat did not directly handle units of measures in most cases. For example, "8 inches" will be implemented as the numeric value 8 with the unit of measure "inches" left to the comment section. In one case, it did convert "7" feet to "7*12" inches but still only kept the numeric value in the code and left the units to the comment section. Yet, based on the code explanations automatically generated, it appeared GPTChat did clearly understand the meaning and the use of units of measures. In addition, the code explanations appear to be smooth natural language without any error (some has period in the end some has not). Another finding was that the GPTChat struggled a little bit in dealing with domain-specific concepts such as the "No. 4" size bars. The Python code generated directly treated "No. 4" size of bars as the numeric value "4". Without domain knowledge in construction, it will be difficult to use this Python code in compliance checking applications.

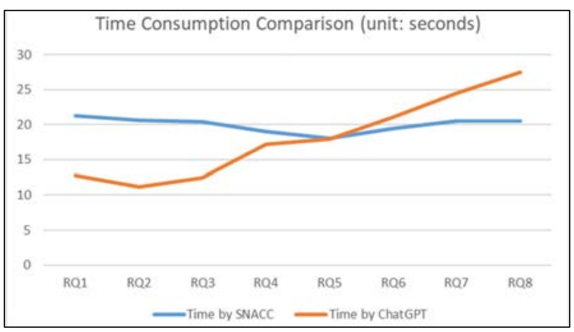

Figure 10. Time Consumption of SNACC and ChatGPT

6 Conclusion

Automated conversion of building code requirements into computable representations is one main barrier to the full automation of building code compliance checking, and thus in turn one main barrier to the wide implementation and scale-up of automated building code compliance checking systems. The state of the art in building code conversion still favored rule-based approach which had better performance than machine learning-based approach. The recent development in deep learning especially the release of GPTChat has a potential to shift that landscape. To investigate if that is the case, the author did a preliminary but systematic set of tests on eight building code requirements with different types and increasing levels of complexity. The most recent semantic NLP-based automated compliance checking (SNACC) system was comparatively evaluated with GPTChat in converting each of the regulatory requirement. SNACC converted requirements into logic rules, whereas GPTChat converted requirements into Python codes. It was found that the processing time curve by SNACC was relatively flat with regard to the complexity of regulatory requirements whereas the processing time by GPTChat stably increased. This could be due to the nature of deep learning behind GPTChat which is inherently more complex than the rule-based algorithm behind SNACC. Furthermore, the Python codes generated did not explicitly treated units of measures, and it will require construction domain knowledge to modify the Python codes to use them in practical building code compliance checking tasks. In conclusion, the GPTChat did not immediately change the landscape in building codes conversion research but it has great potential to facilitate a faster implementation and even scaling up of existing automated code compliance checking systems by reducing the amount of efforts needed in coding. Furthermore, providing specific domain knowledge as part of the prompt could further improve the results, which the author is planning to test in his future work. More holistic testing in the future is also planned to use precision, recall, F1-measure, ROC curve, etc.

References

[1] Hjelseth E. and Nisbet N. Capturing normative constraints by use of the semantic mark-up RASE methodology. Proc., 30th CIB W102, pages 1–10, Queensland University of Technology, Brisbane, Australia, 2011.

[2] Wix J. and Nisbet N. Using constraints to validate and check building information models. Proc., eWork and eBusiness in Architecture, Engineering and Construction, pages 467–476, Sophia Antipolis, France, 2008.

[3] Zhang J. and El-Gohary N. Semantic NLP-based information extraction from construction regulatory documents for automated compliance checking. *J. Comput. Civ. Eng.*, 30(2): 04015014, 2013.

[4] Zhang R. and El-Gohary N. A deep neural network-based method for deep information extraction using transfer learning strategies to support automated compliance checking. *Autom. Constr.* 132(December 2021): 103834, 2021.

[5] Cherpas, C. Natural language processing,

pragmatics, and verbal behavior. *Analysis of Verbal Behavior*, 10: 135–147, 1992.
[6] Ireland C. Alan Turing at 100. On-line: https://news.harvard.edu/gazette/story/2012/09/alan-turing-at-100/, Accessed: 12/01/2023.
[7] Markoff J. Computer Wins on 'Jeopardy!': Trivial, It's Not. Online: https://www.nytimes.com/2011/02/17/science/17jeopardy-watson.html, Accessed: 12/01/2023.
[8] Caldas C.H., Soibelman L. and Han, J. Automated classification of construction project documents. *J. Comput. Civ. Eng.*, 16(4): 234–243, 2002.
[9] Ko T. and Jeong H.D. Syntactic approach to extracting key elements of work modification cause in change-order documents. Proc., Construction Research Congress (CRC) 2020, pages 134–142, Tempe, Arizona, USA, 2020.
[10] Lee J., Yi J.-S. and Son J. Development of automatic-extraction model of poisonous clauses in international construction contracts using rule-based NLP." *J. Comput. Civ. Eng.,* 33(3): 04019003, 2019.
[11] Al-Qady M. and Kandil A. Concept relation extraction from construction documents using natural language processing. *J. Constr. Eng. Manag.*, 136(3): 294–302, 2010.
[12] Moon S., Lee G., Chi S. and Oh H. Automated construction specification review with named entity recognition using natural language processing. *J. Constr. Eng. Manag.*, 147(1): 04020147, 2021.
[13] Akanbi T. and Zhang J. Automated design information extraction from construction specifications to support wood construction cost estimation. Proc., CRC 2020, pages 658–666, Tempe, Arizona, USA, 2020.
[14] Akanbi T. and Zhang J. Design information extraction from construction specifications to support cost estimation. *Autom. Constr.*, 131(November 2021): 103835, 2021.
[15] Ren R. and Zhang J. Semantic rule-based construction procedural information extraction to guide jobsite sensing and monitoring. *J. Comput. Civ. Eng.*, 35(6): 04021026, 2021.
[16] Tixier A.J.P., Hallowell M.R., Rajagopalan B. and Bowman D. Automated content analysis for construction safety: A natural language processing system to extract precursors and outcomes from unstructured injury reports. *Autom. Constr.* 62: 45–56, 2016.
[17] Zou Y., Kiviniemi A. and Jones, S.W. Retrieving similar cases for construction project risk management using natural language processing techniques. *Autom. Constr.*, 80: 66–76, 2017.
[18] Zhang J. and El-Gohary N. Automated information extraction from construction-related regulatory documents for automated compliance checking. Proc., CIB W078 / W102, Sophia Antipolis, France, 2011.
[19] Salama D. and El-Gohary N. Automated compliance checking of construction operation plans using a deontology for the construction domain. *J. Comput. Civ. Eng.*, 27(6): 681-698, 2013.
[20] Salama D. and El-Gohary N. Natural language processing for automated regulatory and contractual document analysis. Proc., 2011 CSCE Annual Conference, Ottawa, ON, Canada, 2011.
[21] Fenves S.J., Gaylord E.H. and Goel S.K. Decision table formulation of the 1969 AISC specification. *Civ. Eng. Studies: Struct. Res. Series*, 347, 1969.
[22] Eastman, C., Lee, J., Jeong, Y., and Lee, J. Automatic rule-based checking of building designs. *Autom. Constr.*, 18(8): 1011-1033, 2009.
[23] Dimyadi, J. and Amor, R. Automated building code compliance checking - where is it at? Proc., CIB World Build. Congress, Brisbane, Australia, 2013.
[24] GPTChat. What is GPTChat? Online: https://chat.openai.com/chat, accessed: 10/01/2023.
[25] Kim S. Writing code snippet using AI – OpenAI ChatGPT. Online: https://medium.com/geekculture/, accessed 11/01/2023.
[26] Doglio F. I asked ChatGPT to build a To-Do app – Have we finally met our replacement? Online: https://blog.bitsrc.io/, accessed 11/01/2023.
[27] OpenAI. CHATGPT: Optimizing language models for dialogue. Online: https://openai.com/blog/chatgpt/, accessed 13/02/2023.
[28] Zhang, J. Automated code compliance checking in the construction domain using semantic natural language processing and logic-based reasoning. Ph.D. Dissertation, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, 2015.
[29] Zhang J. and El-Gohary N. Integrating semantic NLP and logic reasoning into a unified system for fully-automated code checking. *Autom. Constr.*, 73, 45-57, 2017.
[30] Wu J., Xue X. and Zhang, J. invariant signature, logic reasoning, and semantic natural language processing (NLP)-based automated building code compliance checking (I-SNACC) framework." *ITcon,* 28, Special Issue of The Eastman Symposium, 1-18.
[31] International Code Council (ICC). 2009 International Building Code. Online: https://codes.iccsafe.org/content/IBC2009, accessed 01/01/2023.
[32] ICC. 2006 International Building Code. Online: https://codes.iccsafe.org/content/IBC2006, accessed 01/01/2023.