A Hybrid Approach for Automatic Question Generation from Program Codes

Jawad Alshboul, Erika Baksa-Varga
University of Miskolc, Faculty of Mechanical Engineering and Informatics, Miskolc, Hungary

***Abstract*—Generating questions is one of the most challenging tasks in the natural language processing discipline. With the significant emergence of electronic educational platforms like e-learning systems and the large scalability achieved with e-learning, there is an increased urge to generate intelligent and deliberate questions to measure students' understanding. Many works have been done in this field with different techniques; however, most approaches work on extracting questions from text. This research aims to build a model that can conceptualize and generate questions on Python programming language from program codes. Different models are proposed by inserting text and generating questions; however, the challenge is understanding the concepts in the code snippets and linking them to the lessons so that the model can generate relevant and reasonable questions for students. Therefore, the standards applied to measure the results are the code complexity and question validity regarding the questions. The method used to achieve this goal combines the QuestionGenAi framework and ontology based on semantic code conversion. The results produced are questions based on the code snippets provided. The evaluation criteria were code complexity, question validity, and question context. This work has great potential improvement to the e-learning platforms to improve the overall experience for both learners and instructors.**

***Keywords*—Question generation; e-learning; python question generator; semantic code conversion**

I. INTRODUCTION

Automating question generation has become significant with the increasing trend of online learning and its scalability in recent years. Technical courses like learning programming languages are more popular, and there is a massive demand for such subjects. Questions are the primary approach used to evaluate student knowledge [1]. Therefore, creating questions becomes more challenging as the constant growth of e-learning continues, more courses are created, and the pressure on teachers is high. Intelligent and deliberate questions can enhance student understanding and reduce the gap between theory and practice in programming subjects [2]. For example, the article in [3] monitors the performance and behavior of students who engage in courses with self-assessment methods in programming and problem-solving. The research in [4] observes the decentralized practice by monitoring the intensity and timing of the impact on student learning and problem-solving in programming languages. The research paper [5] addresses interactivity while solving problems in programming languages based on learning objects. The article in [6] tries to enhance the use of digital resources for students and instructors. The research papers in [7] and [8] address the learning objects that can be used in different contexts using web3. Finally, the article in [9] suggests collaborative learning to help instructors engage students in generating and evaluating questions. The proposed method focuses on translating Python code into text and uses an AI-based framework to generate questions from the text. We also use ontology to connect and conceptualize the logic of the programming language. Applying ontology ensures interoperability with other systems and reduces the overhead on educational platforms. This work contributes to the e-learning platforms and improves the overall experience for instructors of programming languages. It also enhances the learning path for students who like to learn and do exercises without repeating the same questions. The outcome of this research is to generate meaningful questions based on Python code to assist instructors in creating more questions in a timely manner, thus ensuring students proper learning of the potential programming language. Unlike similar works, most recent research focuses on generating questions from text, while some research focuses on generating questions from visuals or images [10]. This work focuses on generating questions from code snippets using semantic relations to extract the concepts. Generating questions from unconventional sources, such as code snippets, becomes important in providing a better learning experience to large groups of students, especially when dealing with limited information.

A. Research Goal

The main goal of this research is to assist instructors and students in properly evaluating student performance by generating Python-based programming questions from existing materials (i.e., code snippets). The automatic question generation from code snippets will add the possibility of generating a different set of questions based on the same code snippet. Therefore, it leads to a better understanding of the given topic.

B. Research Objectives

To achieve the primary goal of this research, the following objectives are needed:

1) Implement a framework that can interpret Python programming language into text.

2) Enable the framework to comprehend the text and build connections between the programming structures and the semantic concepts.

The rest of the paper is organized as follows: Section II describes related work and some existing approaches. Section III details the question generation framework implemented in this research, and Section IV shows the results. Section V

presents a discussion that summarizes the results. Finally, Section VI concludes the paper and mentions future work.

II. Related Work

The question generation process is a relatively complex and challenging task. It requires adequate experience, high knowledge of the material, and time. With the emergence of online learning, it has become a necessity. The first types of question generation models, such as syntax-based, semantic-based, and other models, started in 2014 [11].

Ontologies are a powerful tool for standardizing knowledge representation, which can be helpful in a wide range of domains, including e-learning. By modeling learning materials with ontologies, it is possible to create more personalized and effective learning experiences, allowing learners to achieve their goals more efficiently [12].

Domain knowledge models can be extremely useful in representing knowledge in a standardized and structured manner, aiding the teaching and learning processes. Python and Owlready2 are used to create the model implementation. Python is a popular programming language for various applications, including machine learning and data analysis. It includes many libraries and frameworks for developing sophisticated software systems. Owlready2 is a Python-based ontology library that simplifies creating and manipulating ontologies in Python code. The researchers created a flexible model that can be used to represent knowledge in a way that can be easily integrated into e-learning systems by implementing the domain knowledge model using Python and Owlready2. It could help develop adaptive learning systems that can tailor the learning experience to the needs of individual students [13].

Despite its importance, implementing question-generation approaches for programming concepts is partially applied in the modern world. Programming languages are an essential topic in computer science and software development, and there is a great demand for effective and efficient ways to teach programming concepts. Developing question-generation approaches for programming languages makes it possible to create many practice questions that can be used to reinforce learning and test understanding [14].

To aid students in their learning, Urazova [15] discusses the development of a system to automatically generate questions regarding UML database design and evaluate student responses. The system generates questions and evaluates student responses using artificial intelligence and methods for natural language processing. The goal is to give students a valuable and practical tool to assess their knowledge of and develop their expertise in UML database design.

A study by Russell in [16] investigated the application of automated code-tracing exercises for teaching introductory programming (CS1) courses. Code tracing is a teaching method in which students mentally run code, follow the control flow, and note the values of variables at each stage. The researchers used an automated system to create code-tracing tasks and assess student responses. The purpose of this system was to serve as a tool for teaching students programming ideas and problem-solving techniques. The researchers evaluated the system's efficacy through studies and student polls and contrasted it with more conventional teaching techniques. The article details the possible advantages of automated code-tracing exercises in CS1 courses and the challenges and limitations that must be addressed.

The use of large language models to automatically provide programming tasks and related code explanations has been used recently. A solution using artificial intelligence was developed to help teachers and instructors construct and deliver efficient programming assignments. A sizable language model was trained on text and computer code to provide workouts and explanations for programming. The model was assessed based on the usefulness and quality of the activities and explanations produced through tests and questionnaires. The promise of leveraging extensive language models for automated programming exercise production is discussed in the study, along with the difficulties and restrictions that must be overcome [17].

Automated programming exercise creation and code explanation have several drawbacks, including bias potential, dependency on large language models, limited capacity to assess student comprehension, significant computing needs, and difficulty in generating high-quality training data. The quality of the language models determines how well the exercises and explanations are created, and there is a chance that the models might be biased. The systems automatically assessing student knowledge could not be reliable and might need a lot of computer power. Producing high-quality training data for large language models is challenging and time-consuming. When utilizing these technologies in educational contexts, these constraints must be taken into account [18].

III. Question Generation Framework

Question generation involves computer understanding of the available materials to propose plausible questions to students. However, two approaches are usually effective: AI-based or semantic-based [12]. The current work uses a combination of both semantic and AI methods to properly generate questions from code snippets based on semantic code conversion. The primary motivation for using the semantic approach is maintaining concept relations in the programming language keywords to increase system intelligence on the programming language rules. Other approaches would not accurately represent the programming language rules, keywords, and concepts. This section will detail the framework architecture, the technology used, and the approach to generate questions.

A. Architecture

To generate questions from existing Python code snippets, an interpreter is needed to dissolve the codes into more understandable concepts. Python or any other programming language is constructed using operators, variables, and functions. Operators such as +,-, AND usually do the actual computing. At the same time, variables are used to store values and recall them with operators to perform specific tasks. Functions contain a list of variables, loops, and operators to be executed in order. The ontology will categorize and conceptualize the list of commands (i.e., variables, operators,

etc.) and the relationships between the concepts in the script. It will build an explained version of the code by processing the code line by line and creating semantic relationships based on the input. Subsequently, the translated code is generated and inserted into an AI question generator called "QuestGen" [19]. This model will generate open-ended questions. Fig. 1 shows the framework data flow and its components.

Awareness of existing technologies and software is essential to construct any framework or software. Such awareness can improve productivity and help address many issues that take a long time. As a result, in this work, we implemented a framework using various third-party software. Table I describes this case's environment settings, tools, and applied libraries. As mentioned earlier, we have used the QuestGen AI model, an open-source NLP library dedicated to creating simple question-generation methods. It is on a mission to become the world's most sophisticated question-generation AI by utilizing cutting-edge transformer models like T5, BERT, and OpenAI GPT-2, among others. The primary objective of QuestGen AI is to simplify the question-generation process, providing support to educators, content creators, and learners in developing educational materials. This tool significantly enhances the efficiency of teaching and learning resource development through automation, ultimately facilitating a more effective educational experience.

Before generating questions, the QuestGen AI model expects a text as input. The ontology mentioned next is responsible for converting the snippet code from the Python programming language into text that humans can understand. Subsequently, this model can generate questions based on the inserted text. The software supports four types of questions, and they are as follows:

- Questions with Several Choices (MCQs)
- Boolean (Yes/No) Questions
- Open-ended Questions
- Question Paraphrase

For the current study, only open-ended questions are considered. Since learning a programming language focuses on understanding the content of a code, it is more suitable to use open-ended questions to assess student knowledge properly.

B. Ontology Design

The ontology is built and compiled using the OWLReady2 library in Python. Such a library would support automating manual activities like adding instances to the ontology. However, the main components and the relationships between concepts should be implemented manually to maintain logical correctness. Translating code into text starts with assigning keywords to ontology classes and describing these keywords. For example, the "=" sign is described in the ontology as an "equal sign", a value of the Assignment subclass in the operator class. The output of the ontology implemented in Python and OwlReady2 is then imported into Protégé for visualization purposes since the visualization is not yet supported on OwlReady2. Fig. 2 shows the ontology design visualization in Protégé.

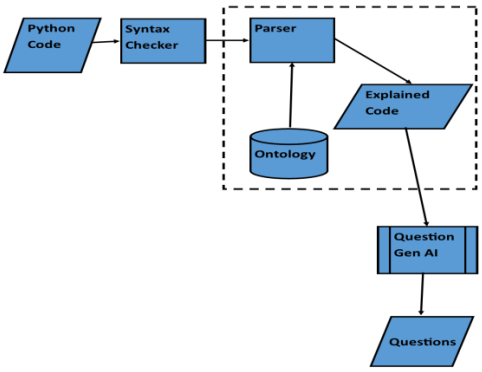

Fig. 1. Proposed framework architecture.

TABLE I. RESEARCH QUESTIONS AND CORRESPONDING RESEARCH OBJECTIVES

Name	Description
OwlReady2	Python library to implement Ontology V 0.37
Protege	Software Application for viewing and modifying ontology
Jupyter Notebook	IDE to develop the framework
QuestGen	AI-based application to generate questions from the text
Python	V 3.11.1

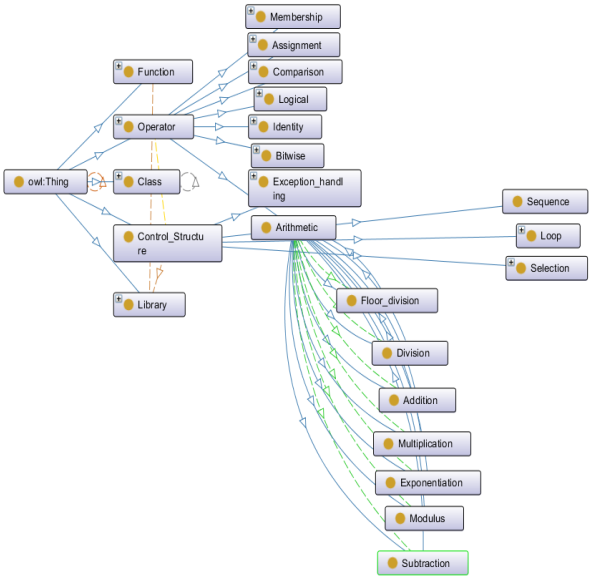

Fig. 2. Ontology design visualization using protégé.

Logical correctness would enforce semantic meaning on the written script. For example, an "elif" statement syntax is valid in Python. However, it cannot exist without having an "if" statement before it. An "elif" should only be coming after an "if". Furthermore, logical correctness would connect all the keywords and describe the semantic relationship between steps. Most essential aspects of Python programming language in the designed ontology are classified as classes and subclasses. For example, in this study, we have categorized the Python language elements and constructs into four main

classes: Control Structure, Function, Library, and Operator. Each subclass of the Operator class contains several instances that would map each instance to the operator class. Such mapping would assist in enforcing the logical correctness of the translated snippet. Fig. 3 shows an instance definition from the constructed ontology. The ontology's capabilities aim to structure the Python programming language to ensure that the computer can collect vocabulary text about the keywords and build sentences based on the combination of the programming language keywords, which can be fed later into the question generation model. The main limitation is that the ontology should be built manually by adding the explanation of all instances, which can be challenging to implement. Further research is needed to improve this approach. Fig. 4 shows a part of the ontology in Python script.

```
<owl:Class rdf:about="#Subtraction">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#has_example"/>
      <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Example
usage of Subtraction</owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#has_description"/>
      <owl:hasValue
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Description of
Subtraction</owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="#Arithmetic"/>
</owl:Class>
```

Fig. 3. Instance definition of subtraction.

```
# Define subsubclasses data structure
subsubclasses = {
    'Arithmetic': ['Addition', 'Subtraction', 'Multiplication', 'Division', 'Modulus', 'Exponentiation', 'Floor division'],
    'Assignment': ['Equal To', 'Add and Assign', 'Subtract and Assign', 'Multiply and Assign', 'Divide and Assign', 'Modulus
    'Comparison': ['Equal', 'Not Equal', 'Less Than', 'Greater Than', 'Less Than or Equal', 'Greater Than or Equal'],
    'Logical': ['and', 'or', 'not'],
    'Identity': ['is', 'is not'],
    'Membership': ['in', 'not in'],
    'Bitwise': ['AND', 'OR', 'XOR', 'NOT', 'Zero fill left shift', 'Signed right shift']
}

# Create operator subclasses and subsubclasses in the new ontology
for operator_type, operator_list in subsubclasses.items():
    with onto:
        operator_type_class = types.new_class(operator_type, (Operator,))
        for operator_name in operator_list:
            operator_name_class = types.new_class(operator_name.replace(" ", "_"), (operator_type_class,))

# ########################################################## FUNCTIONS #####################################
function_sub_classes = ['Built_in', 'Custom_defined']

# Add subclasses for Function
for row in function_sub_classes:
    with onto:
        class_name = row.replace("-", "_")
        NewClass = types.new_class(class_name, (Function,))

# ##########################################################CONTROL STRUCTURE#####################################
control_structure_sub_classes = ['Sequence', 'Selection', 'Loop', 'Exception_handling']
control_structure_selection = ['If', 'Else', 'Elif']
control_structure_loop = ['For', 'While', 'Try']
control_structure_exception_handling = ['Except', 'Finally', 'With']

# Add subclasses for Control_Structure
for row in control_structure_sub_classes:
    with onto:
        class_name = row.replace("-", "_")
        NewClass = types.new_class(class_name, (Control_Structure,))
```

Fig. 4. Ontology in python.

C. Parser

The parser's job is to detach a block of codes into pieces that can match the ontology based on keywords and custom conditions. These conditions are adjusted depending on the inserted snippets. This model uses the ontology to create sentences. It analyzes keywords in the parser and generates sentences explaining the code. For example, a=10, the parser would create “a is a variable. a value is 10”. Fig. 5 illustrates how the code parser algorithm works in the implemented system.

1. **Load the ontology**
2. **Parse Python code and collect keyword explanations**
3. **Split the code into tokens**
4. **Check if a token is a keyword**
5. **Get the explanation of a keyword from the ontology**
6. **Return collected explanations**

Fig. 5. Algorithm steps of the code’s parser explanation.

This parser helps turn Python code (and maybe other types later) into sentences using a set of rules. It maintains whatever logic the ontology possesses about the code. Then, it is fed into the AI model to generate proper questions based on the code interpretation by ontology. The limitation of this parser is that it might struggle with complicated code because it needs specific filters to understand the context and collect the keywords. Fig. 5 describes the steps involved in processing the input and generating results. Initially, the ontology file must be loaded into the environment using an OWL file. Subsequently, the Python source code is provided to the application, where filters extract keywords and retrieve explanations from the defined ontology. Finally, the 'explained code' is passed to the QuestionGenAI framework to generate questions.

D. Question Generation

Over time, there is a growing demand for question generation, a trend that could significantly alleviate the burden on educators and trainers. This is particularly beneficial for scalable learning formats such as online courses. Many models exist for generating questions from regular text; however, understanding code and generating questions from code snippets is not applied due to its complexity. Code-to-text conversion is a challenging task. However, the semantic relationships between the concepts in the ontology are an excellent solution. Fig. 6 shows the whole procedure to translate code into text. In Fig. 6, the code undergoes validation by a parser checker responsible for scrutinizing its syntax. Once the code is confirmed as error-free, the checker directs it to the ontological translator, acting as the parser within our architecture. This parser transforms the code into coherent sentences, forwarding them to the Question Generator AI model to generate reasonable questions. An explanation of the Question Generator AI model is provided in the subsequent section.

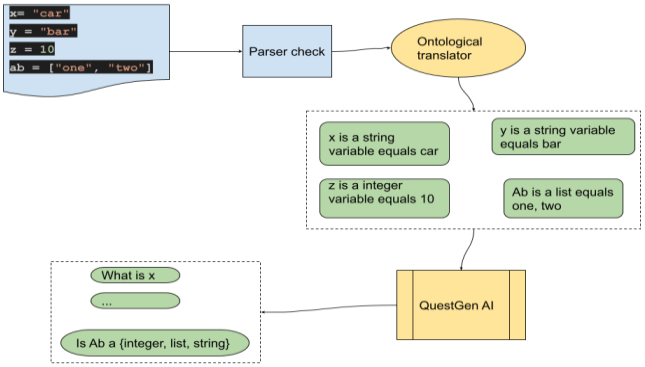

Fig. 6. Question-generation process.

E. QuestionGen AI

The QuestGen AI model is an AI model that can generate questions using AI. The QuestGen project is available in an

open-source format [18]. The model is already trained and can generate high-quality questions based on text fed into the model. Instructors can choose the type of question that can be generated; however, we have only applied open-ended questions. The results summarized in the subsequent section show that the AI model can generate reasonable questions based on the input text and its level of clarity.

- Input: The model can process various types of input, including structured, unstructured, and context-based content such as passages, documents, and articles.
- Field of application: The model is tailored to support the education field across diverse disciplines such as science, history, language arts, and more. However, it does not have the capability to execute or generate programming language code.
- Generation method: It is a semantic-based model designed to comprehend inserted text by leveraging concepts and contextual awareness. This procedure is divided into two main steps. Firstly, it begins with entity recognition, wherein the model extracts crucial information such as dates, names, and relationships, employing part-of-speech tagging. Next, the model applies question templates to the extracted information to match the most suitable predefined question template. To improve question quality, various methods are employed, including probabilistic approaches to refine wording and phrasing within the questions.
- Question format: The model can propose various formats, including open-ended, multiple choice, true/false, and short answer.
- Response format: The responses are generated in both text and JSON formats. Each type of question has its own format. For instance, multiple-choice questions prompt the system to produce the question stem and its corresponding answer choices. This distinction applies to all question types, and the resulting output is tailored accordingly.
- Example: The sentence inserted into the model is “The Industrial Revolution was a period of significant economic and technological change that began in Britain during the late 18th century. It marked the transition from agrarian economies to industrialized ones, with advancements in machinery, transportation, and manufacturing.”
- The generated questions for a true/false type of question are:
    - 'Is the industrial revolution the same as the 'revolution?',
    - 'Was the industrial revolution a period of change?',
    - 'Was the industrial revolution a revolution in the 18th century?'

IV. Results

The results are generated in two versions, one utilizing our proposed model and the other without its use (i.e., by directly inserting the code into the QuestGen AI), as depicted in Fig. 7. The implemented framework facilitates the question-generation process, empowering teachers to automatically generate Python programming language assessment questions for testing students' knowledge. Fig. 8 depicts a straightforward code snippet featuring variable definitions. This figure illustrates specific variables alongside their assigned values, incorporated as a script within the ontology. A Python parser is employed to validate the text as proper code before generating any flawed or erroneous questions to mitigate the potential for incorrect syntax within the inserted code. Fig. 9 displays the translated text derived from the code, providing a textual interpretation for each line. The interpreter presents the variable type and specifies the assigned value for each variable. Fig. 10 showcases the outcomes resulting from inserting the aforementioned text into the QuestGen AI model. Fig. 11 can be seen without having a context. The question generator failed to produce any meaningful questions except for the list variable, where it managed to generate a relevant question. However, the AI model could not comprehend all the lines, hence the presence of the ZERO {} symbol. Fig. 12 exhibits a Python code comprising class and object definitions presented as a string and passed through an ontology to translate it into text. Subsequently, this text is fed into the QuestionGen model to generate questions. In the subsequent examples, only the generated questions and context from QuestGen AI will be showcased, omitting the complete outputs. Moving on to Fig. 13, it explains the preceding code snippet depicted in Fig. 12 using natural language, preparing it for input into the AI generator. Following this, Fig. 14 displays the questions generated from the snippet description, demonstrating the relevance of the generated questions. However, Fig. 15 illustrates the outcome of generating questions without providing a snippet description, resulting in improper questions marked by ZERO{} symbols and inaccuracies. This indicates the necessity of providing a description for accurate question generation. In the third example, depicted in Fig. 16, a function is defined to compute the area of a circle based on its radius. This code incorporates arithmetic operations and utilizes Python's 'math' module. Subsequently, Fig. 17 exhibits the output resulting from describing the aforementioned code to input into the AI model. Meanwhile, Fig. 18 displays the generated questions derived from the description of the code snippet involving mathematical operations. Conversely, Fig. 19 showcases a question generated without describing the snippet. The results depicted in all figures are formatted in JSON, containing both the question and its solution. For open-ended questions, the QuestGen model provides the answer alongside the question, excluding the options. It is worth noting that there are warnings due to deprecated libraries utilized by the QuestionGen model, prompting necessary updates by the authors.

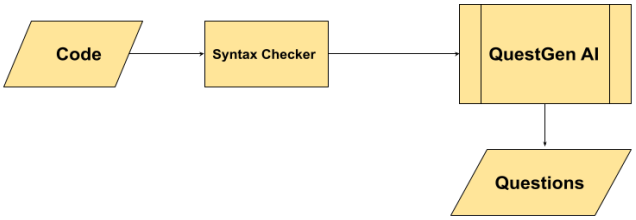

Fig. 7. Generating questions directly from code.

```
# Example Python script to analyze
python_script = """
xfoo = "foo"
ab = ["one", "two"]
cd = ["a boy", 33, "sudden"]
ef = 10
"""
```

Fig. 8. A code snippet with variable definitions.

```
xfoo is a string variable and its value is 'foo'
ab is a list variable and it has 2 items
cd is a list variable and it has 3 items
ef is an integer variable and its value is 10
```

Fig. 9. Generated text from a code snippet.

```
Running model for generation
{'questions': [{'Question': 'What is the value of xfoo?', 'Answer': 'foo', 'id': 1, 'context': "xfoo is a string variable and its value is 'foo'"}]}
{'questions': [{'Answer': 'foo',
                'Question': 'What is the value of xfoo?',
                'context': "xfoo is a string variable and its value is 'foo'",
                'id': 1}],
 'statement': "xfoo is a string variable and its value is 'foo'"}
Running model for generation
{'questions': [{'Question': 'What are the items in the list variable ab?', 'Answer': 'items', 'id': 1, 'context': 'ab is a list variable and it has 2 items'}]}
{'questions': [{'Answer': 'items',
                'Question': 'What are the items in the list variable ab?',
                'context': 'ab is a list variable and it has 2 items',
                'id': 1}],
 'statement': 'ab is a list variable and it has 2 items'}
Running model for generation
{'questions': [{'Question': 'How many items does cd have?', 'Answer': 'items', 'id': 1, 'context': 'cd is a list variable and it has 3 items'}]}
{'questions': [{'Answer': 'items',
                'Question': 'How many items does cd have?',
                'context': 'cd is a list variable and it has 3 items',
                'id': 1}],
 'statement': 'cd is a list variable and it has 3 items'}
Running model for generation
{'questions': [{'Question': 'What is the value of ef?', 'Answer': 'value', 'id': 1, 'context': 'ef is an integer variable and its value is 10'}]}
{'questions': [{'Answer': 'value',
                'Question': 'What is the value of ef?',
                'context': 'ef is an integer variable and its value is 10',
                'id': 1}],
 'statement': 'ef is an integer variable and its value is 10'}
```

Fig. 10. Generated questions for variable definitions.

```
ZERO{}
ZERO{}
Running model for generation
{'questions': [{'Question': 'What is the meaning of cd = "a boy"?', 'Answer': 'boy', 'id': 1, 'context': 'cd = ["a boy", 33, "sudden"]'}]}
{'questions': [{'Answer': 'boy',
                'Question': 'What is the meaning of cd = "a boy"?',
                'context': 'cd = ["a boy", 33, "sudden"]',
                'id': 1}],
 'statement': 'cd = ["a boy", 33, "sudden"]'}
ZERO{}
```

Fig. 11. Generated questions without using the proposed approach.

```
# Example Python script to generate explanations
python_script = """
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

class Student(Person):
    def __init__(self, name, age, school):
        super().__init__(name, age)
        self.school = school

var1 = Person("Jane", 25)
var2 = Student("John", 20, "ABC School")
"""
```

Fig. 12. Python code for defining classes and objects.

```
Person is a class definition
  __init__ is a method
    name is an instance of the property
    age is an instance of the property
Student is a class definition
  __init__ is a method
    school is an instance of the property
  Student inherits from the Person class
var1 is an instance of the Person class with name 'Jane' and age 25
var2 is an instance of the Student class with name 'John', age 20, and school 'ABC School'
Student inherits from Person
```

Fig. 13. Generated explanation of the code in Fig. 12.

```
[{'Question': 'What is person?, 'context': 'Person is a class definition'}]
[{'Question': 'What is __init__?','context': '__init__ is a method'}]}
[{'Question': 'Name is an instance of a property?','context': 'name is an instance of the property'}]
[{'Question': 'What is age an instance of?','context': 'age is an instance of the property'}]
[{'Question': 'What is a student a class definition?','context': 'Student is a class definition'}]
[{'Question': 'What is __init__?','context': '__init__ is a method'}]
[{'Question': 'What is a school?','context': 'school is an instance of the property'}]
[{'Question': 'What class does a student inherit from?','context': 'Student inherits from the Person class'}]
[{'Question': 'What is var1 an instance of?','context': "var1 is an instance of the Person class with name 'Jane' and age 25"}]
[{'Question': "What is the instance of the Student class with name 'John', age 20, and school 'ABC School'?",'context': "var2 is an instance of the Student class with name 'John', age 20, and school 'ABC School'"}]
[{'Question': 'Who does a student inherit from?','context': 'Student inherits from Person'}]
```

Fig. 14. Generated questions for the more advanced snippet.

```
ZERO{}
[{'Question': 'What is the age of the person in def __init__?','context': 'def __init__(self, name, age):'}]
ZERO{}
[{'Question': 'What does age mean?', 'context': 'self.age = age self.age = age'}]
ZERO{}
[{'Question': 'What is the age of the child?','context': 'def __init__(self, name, age, school):'}]
[{'Question': 'What is super().__init__(name, age)?','context': 'super().__init__(name, age)'}]
[{'Question': 'What is self.school?','context': 'self.school = school self.school = school'}]
ZERO{}
[{'Question': 'What is the value of student(John, 20, "ABC School")?','context': 'var2 = Student("John", 20, "ABC School")'}]
```

Fig. 15. Generated questions without using the proposed model.

```
# Define the Python code to be analyzed
python_code = """
import math
def area(radius):
    area = math.pi * radius ** 2
    return area
r = 5
a = area(r)
"""
```

Fig. 16. Code snippet containing a function and arithmetic operations.

```
Imported module: math
area_of_circle is a method definition
rd is a variable
  Its value is Constant(value=5)
ar is a variable
  Its value is Call(func=Name(id='area_of_circle', ctx=Load()), args=[Name(id='r', ctx=Load())], keywords=[])
```

Fig. 17. Generated explanation of the code in Fig. 16.

```
[{'Question': 'What is the name of the module that is imported?', 'context': 'Imported module: math'}]
[{'Question': 'What is a method definition?', 'context': 'area is a method definition'}]
[{'Question': 'What is r?','context': 'r is a variable of type unknown'}]
[{'Question': 'What is Constant(value=5)?','context': 'Its value is Constant(value=5) Its value is Constant(value=5)'}]
[{'Question': 'What is a variable of type unknown?','context': 'a is a variable of type unknown'}]
[{'Question': 'What is the calculated area of the circle?','context': '"a' represents the calculated area of the circle."}]
[{'Question': "What is the value of the call(func=Name(id='area', ctx=Load()), args=[Name(id='r', ctx=Load())?","context': "Its value is Call(func=Name(id='area', ctx=Load()), args=[Name(id='r', ctx=Load())], keywords=[])"}]
```

Fig. 18. Generated questions using the proposed model.

```
ZERO{}
ZERO{}
[{'Question': 'What is the area of the math.pi * radius?', 'context': 'area = math.pi * radius ** 2'}]
ZERO{}
ZERO{}
ZERO{}
```

Fig. 19. Generated question without describing the snippet.

V. Discussion

In this experiment, various code snippets were tested for translation using the proposed ontology and fed into the QuestionGen model to create open-ended questions. Table II outlines the test cases, the generated questions, and the difficulty level of the tested code. It is noticed that human evaluation of AQG results is more accurate than automatic assessments [10]. The validity of the generated code is rated on a scale of 1 to 5, where one represents the least validity and five indicates the highest validity. Difficulty is assessed based on script logic, with five denoting complexity and one representing simplicity. For instance, identifying variable assignments is relatively straightforward, while understanding inheritance is more challenging. Generating appropriate questions from sophisticated or advanced code snippets, such as those utilizing third-party libraries, still presents limitations. Composing accurate questions becomes increasingly tricky as code complexity and inter-line relationships grow. Consequently, further development is necessary to enhance outcomes. Addressing this need will lead to more advanced results. Nevertheless, this study introduces a new dimension to e-learning and supplements existing question-generation approaches that have proven effective in textual sources.

TABLE II. Types of Syntax Covered

Test case	Code level of difficulty	Generated question	Context	Generated question validity
a) Variable declaration	1	What is the value of xfoo?	xfoo is a string variable and its value is 'foo'	4
b) list declaration	2	'What are the items in the list variable ab?	'ab is a list variable and it has 2 items'	5
c) Class declaration	3	What is a person?	Person is a class definition	5
d) Instance and property initialization	4	What is a school an instance of?	'school is an instance of the property'	3
e) Variable initialization, instance initialization, property.	5	'What is var1 an instance of?'	var1 is an instance of the Person class with name 'Jane' and age 25"	4
f) Inheritance identification	5	Who does a student inherit from?	Student inherits from Person	5
g) Libraries import	4	What is the name of the module that is imported?	Imported module: math	4
h) Functions	4	What is a method definition?	area is a method definition	3
i) Variable type	4	What is r?	'r is a variable of type unknown'	4
j) Functions result	5	'What is the calculated area of the circle?	'a' represents the calculated area of the circle.	5

VI. Conclusion

E-learning has become very popular recently, notably accelerated by the onset of the pandemic. One area that has gained considerable attention among researchers is the automatic generation of questions derived from learning materials. However, the predominant focus of existing efforts lies in generating questions from textual content. This work, however, concentrates on generating questions tailored for Python programming language learners derived explicitly from code snippets found in textbooks and course materials. Leveraging ontologies, this approach demands less computational resources, enhancing the scalability of the framework across diverse systems. The proposed framework harnesses ontological mapping, associating each syntactic element with its corresponding meaning and explanation. The process involves translating code into text and subsequently feeding this translated text into an AI-based model for question generation. It aims to alleviate the burden on educators and

reduce the repetition of the same questions for different groups of students. Moreover, the generated questions from code snippets serve to evaluate students' general understanding.

However, the proposed approach still has some limitations. The generation of questions relies solely on the QuestGen AI model, which can occasionally result in poorly phrased questions due to its AI nature. Additionally, the model might struggle to identify certain third-party libraries in complex code snippets. Hence, it represents an opportunity for future work to facilitate the insertion and categorization of concepts from all libraries. Finally, exploring alternative models such as GPT and expanding the framework to recursively process all imported libraries would enable a deeper understanding of complex syntactic structures. This enhancement would empower the ontology to explain code snippets better and generate more nuanced and fitting questions.

ACKNOWLEDGMENT

The authors gratefully acknowledge the financial assistance from the Institute of Information Science, Faculty of Mechanical Engineering and Informatics, University of Miskolc.

REFERENCES

[1] Y. Ham and B. Myers, “Supporting Guided Inquiry with Cooperative Learning in Computer Organization,” in Proceedings of the 50th ACM Technical Symposium on Computer Science Education, Minneapolis, USA: ACM, Feb. 2019, pp. 273–279. doi: https://doi.org/10.1145/3287324.3287355.

[2] R. S. J. d Baker, A. T. Corbett, and V. Aleven, “More Accurate Student Modeling through Contextual Estimation of Slip and Guess Probabilities in Bayesian Knowledge Tracing,” presented at the International Conference on Intelligent Tutoring Systems, in Lecture Notes in Computer Science, vol. 5091. Montreal, Canada: Springer Berlin Heidelberg, Jun. 2008, pp. 406–415. doi: https://doi.org/10.1007/978-3-540-69132-7_44.

[3] C.-Y. Chung and I.-H. Hsiao, “Investigating Patterns of Study Persistence on Self-Assessment Platform of Programming Problem-Solving,” in Proceedings of the 51st ACM Technical Symposium on Computer Science Education, ACM, Feb. 2020, pp. 162–168. doi: https://doi.org/10.1145/3328778.3366827.

[4] C.-Y. Chung, C. Y. C. Edu, and I.-H. Hsiao, “From Detail to Context: Modeling Distributed Practice Intensity and Timing by Multiresolution Signal Analysis,” presented at the 14th International Conference on Educational Data Mining, Virtual: International Educational Data Mining Society, Jul. 2021. [Online]. Available: https://educationaldatamining.org/edm2021/

[5] P. Brusilovsky, M. Yudelson, and I.-H. Hsiao, “Problem Solving Examples as First Class Objects in Educational Digital Libraries: Three Obstacles to Overcome Problem Solving Examples as Interactive Learning Objects for Educational Digital Libraries,” J. Educ. Multimed. Hypermedia, vol. 18, no. 3, pp. 267–288, Jul. 2009.

[6] R. Cafolla, “Project MERLOT: Bringing Peer Review to Web-Based Educational Resources,” J. Inf. Technol. Teach. Educ., vol. 14, no. 2, Apr. 2006.

[7] H. K. M. Al-Chalabi, “Evaluation of a Multi-Parameter E-learning System using Web 3.0 Technologies,” presented at the 13th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), Pitesti, Romania: IEEE, Jul. 2021, pp. 1–4. doi: https://doi.org/10.1109/ECAI52376.2021.9515191.

[8] H. K. M. Al-Chalabi and U. C. Apoki, “A Semantic Approach to Multi-parameter Personalisation of E-Learning Systems,” presented at the International Conference on Modelling and Development of Intelligent Systems, in Communications in Computer and Information Science, vol. 1341. Sibiu, Romania: Springer International Publishing, 2021, pp. 381–393. doi: https://doi.org/10.1007/978-3-030-68527-0_24.

[9] P. Denny, A. Luxton-Reilly, and J. Hamer, “The PeerWise System of Student Contributed Assessment Questions,” in Proceedings of the tenth conference on Australasian computing education, Wollongong, Australia, Jan. 2008, pp. 69–74. doi: https://dl.acm.org/doi/10.5555/1379249.1379255.

[10] N. Mulla and P. Gharpure, “Automatic Question Generation: A Review of Methodologies, Datasets, Evaluation Metrics, and Applications,” Prog. Artif. Intell., vol. 12, no. 1, pp. 1–32, Jan. 2023, doi: https://doi.org/10.1007/s13748-023-00295-9.

[11] G. Kurdi, J. Leo, B. Parsia, U. Sattler, and S. Al-Emari, “A Systematic Review of Automatic Question Generation for Educational Purposes,” Int. J. Artif. Intell. Educ., vol. 30, no. 1, pp. 121–204, Mar. 2020, doi: 10.1007/s40593-019-00186-y.

[12] J. Alshboul and E. Baksa-Varga, "A Review of Automatic Question Generation in Teaching Programming," Sci. Inf. Organ., vol. 13, no. 10, pp. 45–51, 2022, doi: 10.14569/IJACSA.2022.0131006.

[13] H. A. A. Ghanim, J. Alshboul, and L. Kovács, “Development of Ontology-based Domain Knowledge Model for IT Domain in E-Tutor Systems,” Int. J. Adv. Comput. Sci. Appl., vol. 13, no. 5, pp. 28–34, 2022, doi: https://dx.doi.org/10.14569/IJACSA.2022.0130505.

[14] J. Alshboul, H. A. A. Ghanim, and E. Baksa-Varga, “Semantic Modeling for Learning Materials in E-Tutors Systems,” J. Softw. Eng. Intell. Syst., vol. 6, no. 2, pp. 85–91, Aug. 2021.

[15] T. Urazova, “Building a System for Automated Question Generation and Evaluation to Assist Students Learning UML Database Design,” University of British Columbia, 2022. [Online]. Available: https://open.library.ubc.ca/soa/cIRcle/collections/undergraduateresearch/52966/items/1.0413656

[16] S. Russell, “Automated Code Tracing Exercises for CS1,” presented at the Computing Education Practice 2022, Durham, United Kingdom: ACM, Jan. 2022, pp. 13–16. doi: https://doi.org/10.1145/3498343.3498347.

[17] S. Sarsa, P. Denny, A. Hellas, and J. Leinonen, “Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models,” presented at the International Computing Education Research, Lugano, Switzerland: ACM, Aug. 2022, pp. 27–43. doi: https://doi.org/10.1145/3501385.3543957.

[18] M. Sh. Murtazina and T. V. Avdeenko, “The Constructing of Cognitive Functions Ontology,” presented at the 14th International Symposium "Intelligent Systems, Moscow, Russia: Procedia Computer Science, 2021, pp. 595–602. doi: https://doi.org/10.1016/j.procs.2021.04.181.

[19] R. G. Golla, V. Tiwari, P. Chokhra, and H. Okada, “QuestGen AI.” [Online]. Available: https://github.com/ramsrigouthamg/ Questgen.ai.