NLI-GSC: A Natural Language Interface for Generating SourceCode

Aaqib Ahmed R.H. Ansari[1]
Vidyalankar Institute of Technology
University of Mumbai
Mumbai, India

Dr. Deepali R. Vora[2]
Symbiosis Institute of Technology
Symbiosis International University
Pune, India

***Abstract*—There are many different programming languages and each programming language has its own structure or way of writing the code, it becomes difficult to learn and frequently switch between different programming languages. Due to this reason, a person working with multiple programming languages needs to look at documentations frequently which costs time and effort. In the past few years, there have been significant increase in the amount of papers published on this topic, each providing a unique solution to this problem. Many of these papers are based on applying NLP concepts in unique configuration to get the desired results. Some have used AI along with NLP to train the system to generate source-code in specific language, and some have trained the AI directly without pre-processing the dataset with NLP. All of these papers face two problems: a lack of proper dataset for this particular application and each paper can convent natural language into only one specified programming language source-code. This proposed system shows that a language independent solution is a feasible alternate for writing source-code without having full knowledge about a programming language. The proposed system uses Natural Language Processing to convert Natural Language into programming language-independent pseudo code using custom Named Entity Recognition and save it in XML (eXtensible Markup Language) format which is an intermediate step. Then, using traditional programming, this system converts the generated pseudo code into programming language-dependent source-code. In this paper, another novel method has been proposed to create dataset from scratch using predefined structure that is filled with predefined keywords creating unique combination of training dataset.**

Keywords*—*Natural Language Processing (NLP); Natural Language Interface (NLI); Entity Recognition (ER); Artificial Intelligence (AI); source code generation; pseudocode generation

I. Introduction

Source-code is a list of human-readable instructions written in particular programming language. The aim of source-code is to check for precise specification, format and rules so that it can be interpreted into machine language [1]. Therefore, source-codes are the fundamentals of a computer program. It is usually written by a programmer or developer that has some training and knowledge of the programming language. The are many independent languages and each has its own distinctive way to writing instructions.

Natural Language Interface (NLI) provides a different input method in which users can interact with computer using spoken human language, like English instead of using a graphical user interface (GUI), command line interface (CLI) or computer languages like C and Python [2]. NLI enables the computer to recognize and understand the flow of human language by providing an abstract layer that connects computers to users [3]. It enables users to enter their search queries in natural language which can be in either spoken audio or written text. The goal for most natural language systems is to make the system easier to use and to provide an interface that decrease the training time required for users.

The proposed system aims to generate source code of various programming languages like python and C using natural language as input. Making use of NLI to generate source-code can help beginners understand the language well. It can also help professionals to increase their working speed as uncommon and easy problems can be solved without going through documentation of that programming language.

As writing source-code is becoming more widespread and complicated, it is becoming an essential to automate writing simpler source-code by AIs to save time in writing, learning and understanding source-code. With Natural Language Processing (NLP) getting better and simple with each year and the demand for writing new and complex source code is increasing every year, it was inevitable for these two fields to join. There are many natural language interfaces that connect NLP with databases but not many programs that connect NLP with computer languages. Creating an Natural Language Interface that helps programmers to create source code will reduce the time it takes for them to write source code as they will refer to complex documentation less frequently and will reduce the bar to enter the world of programming language.

II. Literary Study

A. Different Natural Language Interface Approaches

Review of different approaches in natural language interfaces to databases [4] published by Reshma E. U. and Remya P. C. explores some Natural Language Interfaces to Databases (NLIDB) trends in 2017. They found that current NLIDB system consists of following types: 'Pattern matching' (i.e. Using manually defined rules), 'Syntax based system' (i.e. Creating parse tree and mapping it to database), 'Semantic grammar system' (i.e. Passing user input with hard wired semantic grammar and then creating parse tree which will be mapped to database) or 'Intermediate representation system' (i.e. it first translates the natural language input into intermediate logical query and then, it translates intermediate logical query into database query language.

There are many such systems that follow the same techniques and patterns with minor changes, namely, "IQS - Intelligent Querying System using Natural Language Processing" [5] and "MyNLIDB: A Natural Language Interface to Database" [6].

The advantages of these NLIDBs are that the users does not need to learn any artificial language, no need for spending time on training, simple and easy to use, are better for some questions and have high fault tolerance.

The disadvantages of these systems are that they deal with small amount of natural language i.e. can recognize limited set of words, errors and failures are not properly handled, ambiguity i.e. one word having many unrelated meaning can cause the query to change its meaning and finally users may not construct the query using recommended or pre-programmed words.

B. Natural Language Query to SQL

In the paper "Formation of SQL from Natural Language Query using NLP" M. Uma et. al. [7] used the following techniques to extract information from natural language.

First, they extracted 'attribute' using Parts Of Speech (POS) tags. They used tokens next to a proper noun (i.e. NNP tag in NLTK Library). To search for 'date' they used regular expression (RegEx) to extract it using common writing formats. To extract 'fares' the used lemmatized word 'fare' and finally they used RegEx again for extracting train names.

This system is very rigid and can only perform SQL tasks on predefined database. But, we can use these methods to tag our own data to give to an AI model

C. Conversion of Natural Language Query to SQL Query

In the paper titled "Conversion of Natural Language Query to SQL Query" by Abhilasha Kate et. al. [8] they first performed "Tokenization" on their input sentence and remove the stop words, then those tokens would be passed onto "Lexical analysis" which will replace all the words with their dictionary counterpart. This is the step where the natural language starts to look like a SQL sentence. Lastly, "Semantic Analysis" is performed which will replace natural sentence (e.g. less than or equal to) to their symbol counterparts (i.e. $<=$).

The given system has a shortcoming in lexical analysis phase as all the keywords needs to be known beforehand to be able to match those keywords with dictionary.

D. Language to Code

The paper titled "Language to Code with Open Source Software" [9] published by Lei Tang, Xiaoguang Mao and Zhuo Zhang uses an encoder-decoder technique to automatically train NLP to generate source-code. They first converted the natural language descriptions into word-embeddings and fed it into the encoder to generate coding vector. Then the decoder maps this vector back into the desired code. They used LSTM neural network to train their model. Due to the labor-intensive nature of generating dataset they used a previously proposed method by Gu Xiaodong et.al in the paper "Deep code search" [10]. In this paper they proposed a unique method to create training dataset i.e. they used comments from a Java program and its attached code snippet from open source Java projects as the dataset.

Even though this technique covers many complicated code scenarios, this technique is limited by the programming language it generates (here they can only generate Java source-code). To generate source-code in other programming languages, we need to create another database from scratch which is a lengthy and tedious task. From this system we can adopt the training methods they used with the databases tagging techniques they used.

E. Natural Language Database Query Interface

"A Simple Guide to Implement Data Retrieval through Natural Language Database Query Interface (NLDQ)" [11] published by Tameem Ahmad and Nesar Ahmad uses a straight forward approach to convert a natural language statement to database query. They first used "Tokenizer" to divide the input into individual tokens or words. They then used "Parsing" to create a parse tree with its related POS (parts-Of-Speech) tags. After this they used "Syntactic Comparison" to check if any keywords that appeared in the input is already available in the database as either a direct match or a alias of it. Lastly, in "SQL Generator" they used static templates that can be used as fill-in-the-blanks to choose the correct template and fill all the related fields labeled in previous steps.

The main advantage of this type of system is that it is easy to make any new changes that the client requests. But the downsides are that this is a very rigid system that is tied to a particular database. Although, we won't tie our system with templates, but each programming language uses a predefined format (i.e. main function, indentation, parentheses "}", etc.) which we can add to our system.

F. Modified Co-occurrence Matrix Technique

Anuradha Mohite and Varunakshi Bhojane [12] proposed a updated way to find co-occurrence matrix formulation in the paper titled "Natural Language Interface to Database Using Modified Co-occurrence Matrix Technique". They first parsed the input to create POS (i.e. Parts Of Speech) tags and parse tree. They then used modified Hyperspace Analogue to Language (HAL) matrix to find tokens related to nouns. Using cosine-similarity they get a table with nouns associated with different POS and using this technique they identified the WHERE clause in SQL (Structured Query Language).

With the hardest part of the query now identified they used stemming technique to convert all words into their common roots and then used semantic mapping to find words such as min, max, avg, $>=$, etc. Once these word are identified they used bi-gram algorithm to find correct attribute and table name from the database and create the final query to be displayed to user along with its output. "Natural Language to Structured Query Language using Elasticsearch for descriptive columns" [13] uses similar approach with changes made in word embeddings.

Their clever use of POS tag pair such as "numeric value-noun pair" or "proper noun-noun pair" to find where clause in SQL can help us to identify inconsistent attributes such as names of variables and functions in our system.

G. Pseudocode to Source-code

Teduh Dirgahayu et.al. [14] proposed a method to automatically convert pseudocode to Source-code. In this method, the pseudocode is first translated to an intermediate model then to source-code. The intermediate model consists of a parse tree that is created with the help of a tool called ANTLR. This represents pseudocode in a more structured and language independent way. Then language dependent tool for generating source-code is created.

The main shortcoming of this method is that the pseudocode which is in the form of XML needs to be created manually. This manual pseudocode and source-code must comply to their respective grammars (i.e. metamodels). The XML intermediate model must comply to a XML schema.

H. An XML-based Pseudo-code Online Editing and Conversion System

Liu Haowen et.al in their paper "An XML-based Pseudocode Online Editing and Conversion System" [15] introduced a innovative way to convert pseudo code into source-code. They first convert the input pseudo code into an XML (eXtensive Markup Language) format using DOM4J package available in Java programming language. From there, using the same DOM4J package, they converted the pseudo code in XML format into Java source-code. In doing so, they also created XML tags for pseudo code which we can use in our system as a reference.

The reason to choose XML is that it is a cross-platform language i.e. they can be used in any software and in any operating system making this system independent of any programming language.

I. Pseudocode to Source-code using NLP

Ayad Tareq Imam et.al [16] proposed a unique way to convert pseudocode to source-code i.e. using NLP. The problem with this method is that it presents a complex solution to a simple problem. As seen in previous 2 papers [14] & [15], the same output can be achieved with simple one-to-one mapping.

J. Generating Source-code without NLP

Till here we have studied ways to generate structured code like SQL queries (due to lack of source-code generation) using NLP techniques. There are other papers that can generate proper source-code but they bypass the NLP requirements and directly train their models to generate source-code from training data.

Some such papers are "DeepCoder" [17], "Text2App" [18], "Generating Pseudo-Code from Source Code Using Deep Learning" [19], "Incorporating External Knowledge through Pre-training for Natural Language to Code Generation" [20], "In-IDE Code Generation from Natural Language" [21] and "Programming with a Differentiable Forth Interpreter" [22].

All these papers faces a problem of lack of source-code datasets. Some uses premade dataset, many generate their own. Another problem they face is all these papers generate source-code in one specific programming language

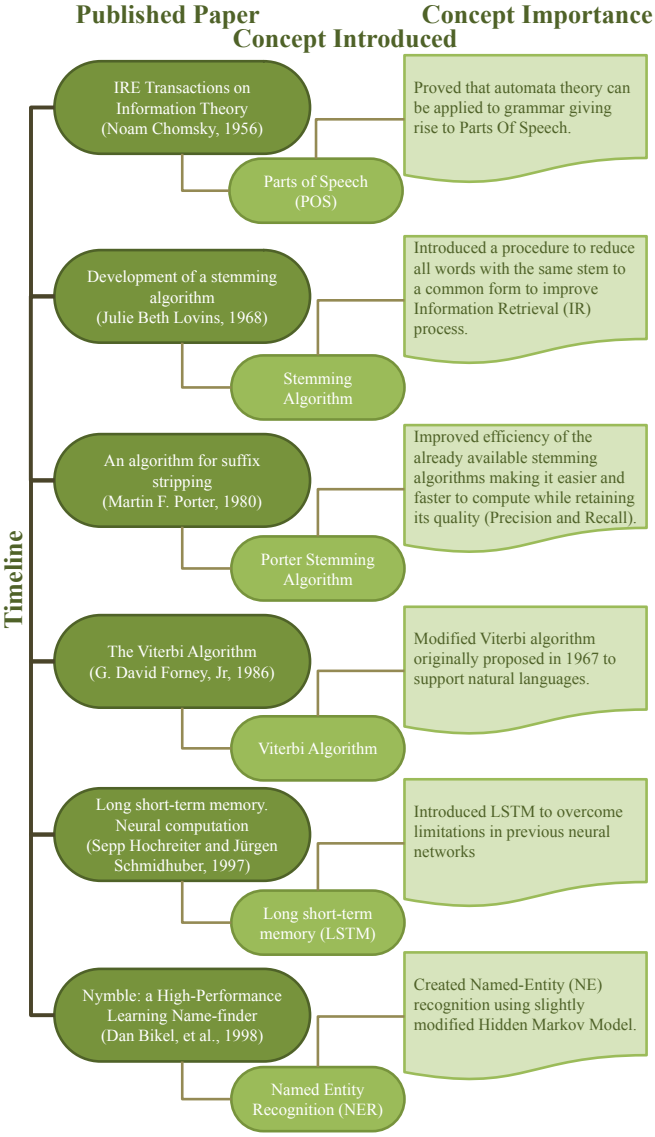

Fig. 1. History of NLP Concepts.

K. History of Various Concepts of NLP

Fig. 1 lists the papers where important concepts have been introduced in history of NLP with the importance of the paper. These concepts have been used in our system. The Long Short-Term Memory (LSTM) neural network [23] has not been mentioned explicitly in our system but we used this neural network to train our various NLP models.

Some may argue that the NLP as a field began when Warren Weaver mentioned using of modern computing devices to translate from one language to another in his memorandum [24] in 1949 and the rest is history.

1) Parts of Speech: Noam Chomsky in 1956 [25] introduced the concept that grammar of a language can be viewed as a theory of the structure of this language and is based on certain finite set of observations. It introduced the finite-state language for NLP that we called today as Parts Of Speech (POS).

2) Stemming Algorithm: The stemming algorithm which converts all the redundant words to its common "stem" was published in 1968 by Julie Beth Lovins [26]. Surprisingly its main purpose was not for NLP purposes but for retrieving huge amount of data from databases (back then data transfers were a lot slower).

3) Porter-Stemming Algorithm: The famous porter-stemming algorithm was easy to find as it is in public repository online. This was develop to further improve the efficiency of the information retrieval systems of the various stemming algorithms available at that time [27].

4) The Viterbi Algorithm: Although, Viterbi algorithm was originally proposed to calculate the error bounds in convolutional codes [28] as just a proof of concept, David Forney Jr. [29] in his paper "The Viterbi Algorithm" modified this algorithm for NLP purposes. Till this date we use this algorithm alongside with Hidden Markov Model to predict the POS tag of a word with probability even though an AI solution exists.

5) Named Entity Recognition (NER): "Nymble: a High-Performance Learning Name-finder" published by Daniel Bikei et.al. [30] was the first to introduce a Named-Entity Recognition system using slightly modified Hidden Markov Model. It performs at or above the 90% accuracy level, often considered "near-human performance".

Observing all those systems, we came to conclusion that there is a lack of NLP system where a natural language is converted into a source-code (all those systems converted natural language into database queries). Also, those systems that do convert to natural language does not allow customizations (i.e. adding custom keywords to recognize) as a feature.

Our project intends to introduce a novel method that helps the programmers in developing source-code and increasing their speed and efficiency.

III. Problem Statement

The project intends to introduce a novel method that helps the programmers in developing source-code increasing their speed and efficiency.

- An interface to take natural language as input.
- An option to select programming language on the interface (initially python).
- Convert natural language into pseudo code using NLP (using intent recognition and entity recognition).
- Convert pseudo code into source-code (using traditional programming).
- Output the source-code onto the interface.

IV. Problem Definition

For a programmer, the ability to learn a programming language's commands and functions on the fly is crucial to the time they spend on creating a source-code. No matter the skill of the programmer, there are always are cases where they have to tackle uncommon features present in a programming language which they have not seen before. To solve those uncommon and unseen features they refer documentation of that programming language and then learn from it which requires time.

The programmers can define the problem in natural languages easily, the problem lies in remembering the exact keywords and parameters. By providing a Natural Language Interface we can shorten the time the programmer takes to search through documentations. It can also help to reduce the skill and experience required to write complicated source-code which heavily depends on using functions.

V. Proposed Work

1) A python based interface to input English natural language and an option to chose target programming language.
2) Gather training data from the internet. The data will be simple beginner's problem statements from various sites for variety.
3) Provide "Parts Of Speech" (POS) tags to each word.
4) Train "Intent Classification" to recognize different operations such as "addition", "multiplication", "variable declaration", "print statement", etc.
5) Train "Entity Recognition" to recognize various data types of variables.
6) Identify variable names through POS if they are present in the input.
7) Write a function for each "intent" and extract information like number of variables, their type and name if it is present and other information if required.
8) Convert the extracted information into a pseudo code in a XML format.
9) Read that XML file and convert it into source-code through python.
10) Save the source-code into a file.
11) Display the file into the interface.

VI. Libraries Required

A. TKinter

Tkinter is the standard GUI library for Python. Tkinter provides a fast and easy way to create GUI applications [31]. Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

B. NLTK vs SpaCy

NLTK (Natural Language ToolKit) [32], spaCy [33] and Stanford's CoreNLP [34] are all similar libraries that provide off-the-shelf functions for NLP. As we are using "python" programming language for creating a demo, we need to list all pros and cons on NLTK and spaCy libraries as those support the python language, coreNLP does not (it only supports Java). Table I lists the differences between NLTK and spaCy libraries.

The paper titled "Using Natural Language Processing to Detect Privacy Violations in Online Contracts" by P. Silva et.al. [35] and another paper titled "Extractive Automatic Text Summarization using SpaCy in Python & NLP" [36] made comparison between spaCy, coreNLP and NLTK and came to conclusion that coreNLP has the best performance in terms of precision, recall and F1 score followed by spaCy and lastly NLTK.

TABLE I. NLTK LIBRARY VS SPACY LIBRARY

NLTK	SpaCy
NLTK (Natural Language TookKit) library was released in 2001.	SpaCy library was released in February 2015.
NLTK library is only supported in "python" programming language.	SpaCy is supported in number of programming languages like "R", "ruby", "cpp", "java script", ".net", "python", etc.
NLTK has a lot of algorithms as compared to spaCy which is helpful in learning and research applications.	SpaCy has a small group of algorithms that they regularly update, which is critical in industry usage.
As it is not updated regularly, the algorithms provided by NLTK are comparatively slower than spaCy.	With their regular update to latest techniques spaCy is significantly faster than NLTK.
NLTK incorporates several languages.	SpaCy have statistical models for seven languages including English, German, Spanish, French, Portuguese, Italian, and Dutch, It also braces "named entities" for multi-language.
NLTK is string processing library. It takes input as strings and provides output as string or lists of strings.	SpaCy uses object-oriented approach. It takes input in string but return the output in objects.
NLTK does not support word vectors.	SpaCy has support for word vectors.
NLTK tries to break the text into sentences. It just returns the words itself, no extra information is given with it.	SpaCy builds a semantic tree for individual sentence as higher a potent approach, returns more information.
NLTK has inferior precision, recall and F1 score than spaCy.	SpaCy has better precision, recall and F1 score than NLTK.

C. Regex

Although spaCy provides "Pattern Matcher" functionality to its toolbox which is similar to regex, a regular expression or regex can be used in some limitations of spaCy's pattern matcher.

A regex defines a set of strings that lets you check if a particular string is present in a large text [37]. The most common usage of regex is to alert system administrators if an error appeared in log files. Another example would be to extract phone numbers, email addresses from a large database. It is commonly used where fixed pattern appears.

To find a email address that has the format "someone@somedomain.sometopleveldomain" (eg. john@gmail.com) we use the regex "$[a-zA-Z0-9_+-]+@[a-zA-Z0-9-]+.[a-zA-Z0-9-.]+$" to find emails in plethora of texts.

VII. WORKING MODEL

The main goal of our system is to convert "Natural Language" statements into source-code. It is able to handle one line statements as input and can generate 3-5 lines source-code depending on the input given.

Fig. 2 shows the working of our system when the system has been deployed at the client side. It is a step-wise working model which defines the processes from user input to showing output to the user and all the other steps in-between. In this figure, the light-blue nodes denotes the generation of files. For example, the node "Pseudo code statement/command" denotes saving the .xml file in log folder. The white nodes denotes processing.

A. NLP Translator

The first step in this process is to get the user input. The input has to be in English natural language and has to describe the programming problem. Once the input is received the "NLP Translator" uses a trained NLP model to break down the input and extract important information from it. In this process, the first thing applied is "Tokenizer" to split the input into individual tokens or words, then entity extractor identifies important keywords like equals sign, condition statements, etc. Next, the "Intent Identifier" identifies what kind of operation does the user wants to perform like "addition", branching, etc. Lastly, POS is used to identify the variable name, its value (if present) and function/program name.

B. Post Processor

Once all the input has been identified, it's time to convert the extracted information into a pseudo code, this is done is "Post Processing" step. It first arranges the input in specific order that resembles a pseudo code, adds extra information if missing and the using XML parsers converts the pseudo code into XML. After this we are left with pseudo code file that is easier to read by both humans and computers thanks to the XML format.

C. Rule Based Translator and Its Post Processor

Rule Based Translator is an easier step that reads the pseudo code line by line and converts the keywords from universally understood to language specific format. The post processor adds extra features to the syntactical code like parenthesis "{}", indentation, semi-comma ";", etc. The output is then saved on disk and displayed to the user on the interface.

VIII. ALGORITHM

Algorithm given in Fig. 3 is a coarse-grain view of the proposed system that gives the basic steps needed to implement this system.

We first start by creating an interface that can get input and provide output. Second step is to search for a dataset. Since, we did not find any we created the dataset from scratch. The method of which is defined further ahead.

Next is to create a Named Entity Recognition or NER model. To create a NER model we first need to provide Parts Of Speech or POS tags which then be used by NER model to identify various entities.

Once the system is trained we then start extracting information from it. The first information we extract is variables, their name, datatype, scope and value. After which "operations" are identified and all the variables associated with it are processed i.e. each individual part of the operation is identified (for example, in the statement "add var1 and var2", add is the type of operation, var1 and var2 are the two variables associated with it).

The extracted information is then parsed into XML format and a pseudocode file is generated. Then this XML pseudocode file is read again by the system and is converted into source-code. This source-code is saved on the user's system and displayed on the interface.

IX. SYSTEM DESIGN

Fig. 4 is a detailed view of the proposed system that explains the steps needed to create this system.

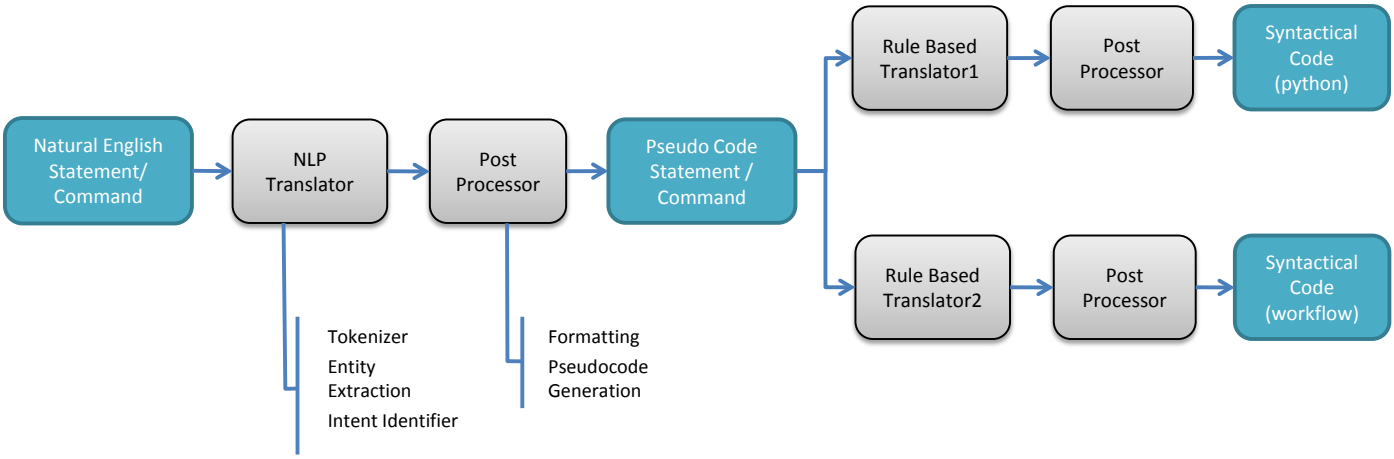

Fig. 2. NLI-GSC Working Model.

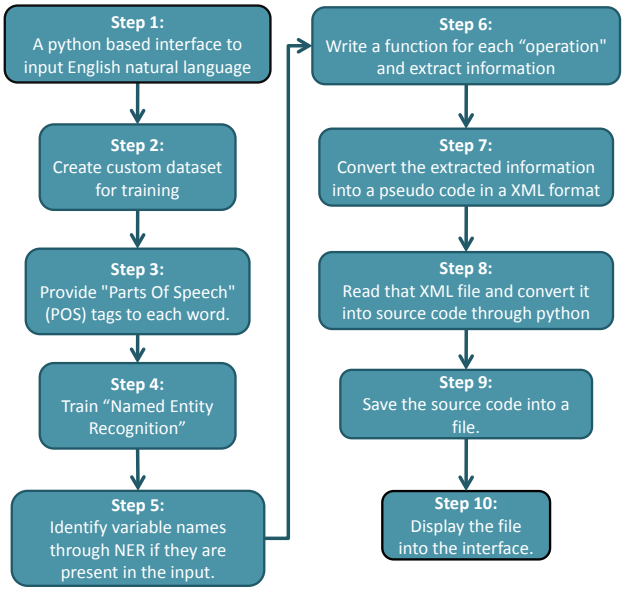

Fig. 3. Algorithm of NLI-GSC System.

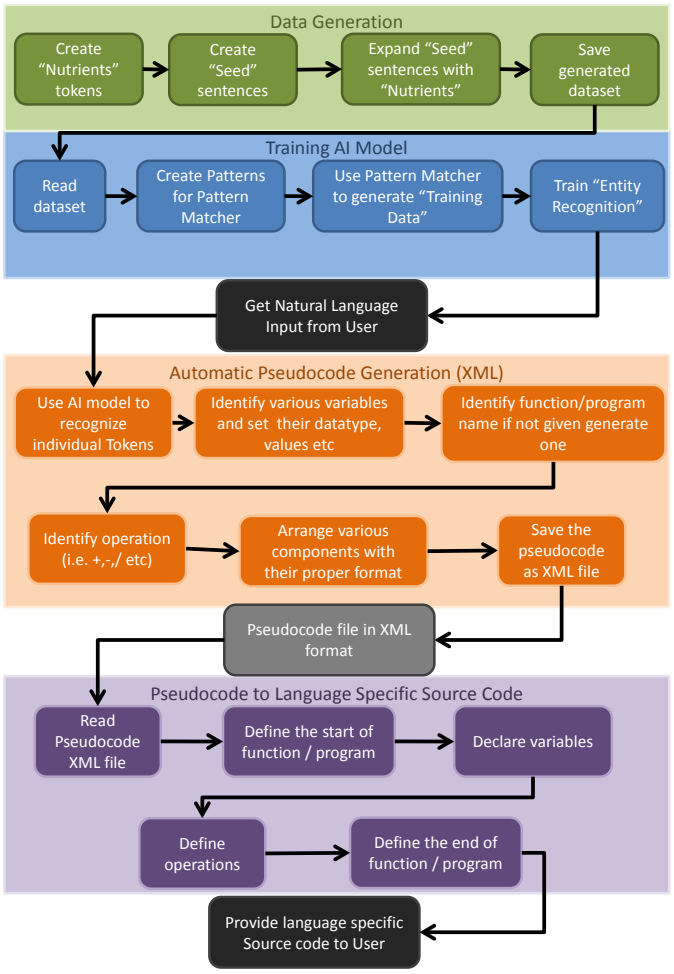

Fig. 4. NLI-GSC System Design.

A. Data Generation

There are not many NLP datasets, if any, that address the programming aspects that we can use. Generating data manually for an AI system is very tedious and time-consuming process as it requires a minimum of hundreds of data to provide any acceptable results. Hence, due to these reasons we have developed an automated system that can create its own dataset.

As seen in Fig. 4 & 5, we first define ”nutrients” tokens i.e. variations of similar word. For example, ”DATATYPE” is a nutrient containing integer, int, float, str, string, etc. We then define ”seed” sentences. Seed sentences are made up of combinations of ”nutrients”. An example of seed sentence is ”COMMAND DATATYPE VARIABLE” which when expanded will result in sentences like ”create integer var1”, ”define float area” and ”initialize string text_1”.

The next step is to expand the seeds using individual nutrient tokens. This will result in generating all the possible combination of statements for the given format. In our program, we were able to generate 1,565,668 statements from 23 seed statements (each seed’s length ranging from 3-6 nutrient

tokens) and 19 individual nutrient tokens. Since, 1.5 mil is too much of a dataset, we have reduced it with the ratio 1:20 resulting in getting a final dataset of 78,284 statements.

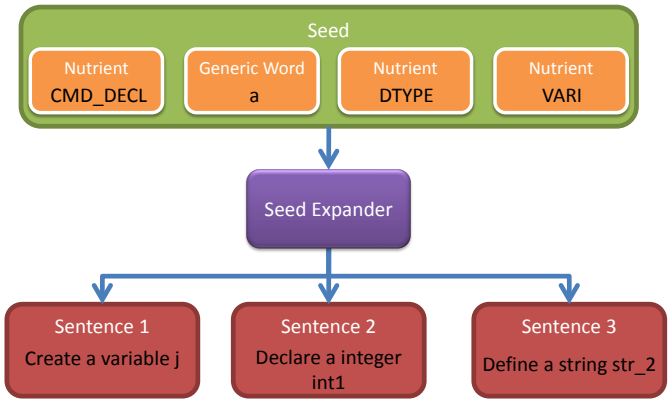

Fig. 5. Data Generation Process.

B. Training AI Model

For training of the AI model, we used "spaCy" library [33]. Unlike its NLTK counterpart which focuses on academics and learning, spaCy is built for industrial uses and production.

We used Named Entity Recognition (NER) as our model to tag words that may belong to certain instruction. For example, the word "int" will tell NER model that it belongs to "datatype" while words like "addition", "sub", "*" will be recognized as "operations".

In training in the AI model, we first need to create a dataset from data which we have achieved using spaCy's "Pattern Matcher". Since, we can control which words appear in the data in Data Generation phase, we can easily use pattern matcher to find a word or sequence of words to tag it. Once all the relevant words are tagged and a training dataset is created we can proceed to train it. Again, we used spaCy standard libraries to train. We used 3 epochs with batch size = 100 and got the final training loss of 1.95.

C. Automatic Pseudocode Generation

When the user inputs their natural language query, our system automatically converts it to pseudocode and saves it in XML format. The process is as follows.

After the user has given their input, the system uses NER model trained in previous step to identify various tags examples can be seen in Fig. 6 and 7. Once the tokens have been recognized, we lemmatize them for easy recognition. We lemmatize all the types of tokens inside the tags except variables tag, since they are user given names. We process them based on the tags it has been assigned.

The variables are assigned names (if none is given it will automatically generate based on datatype), datatypes (If none is given, it will first try to get it from another variable in the input if not it will be assigned as string.), scope (local or global) and values (if they are given). If the program recognizes that it does not require any variables it will skip this step.

The input is recognized as either a function or a program and then the name of the function/program is recognized if the user has mentioned it in the query. A name is automatically generated based on available tags if the user has not mentioned the name explicitly. Once an operation tag is detected it will take all the variables associated and arrange them in specific pattern (i.e. $ans = var1 + var2;$).

Finally, all the components will be arranged by the order of appearance. First, the program/function will be declared with a proper indentation, then variables will be declared and then operations will be written and lastly if the language demands, the program/function will end (For example, some program requires "}" to end them). All this will then be parsed to XML file and saved on the desired location.

D. Pseudocode to Language Specific Source-code

Once the pseudocode file is ready, the only thing left to do is to convert it into a programming language. Using techniques that convert XML to program [14] [15], we can easily convert pseudocode to source-code. An automatic pseudocode generation is independent in creating a language specific source-code. Hence, we can- create the output of any programming language by creating a new file.

It's a simple process that reads the XML from top to bottom and depending on the tag encountered it will use a specific format to fill in the blanks.

X. Results

Our system produces multiple outputs before reaching the final source-code output. Below are the outputs according to the order they are generated.

A. Generating Data for Training

As explained in Section IX-A, seed statements are used to generate input data that is later used to train the NLP model. Table II lists all the seed statements used in the system along with some sample output it generates. This output is unlabelled text data that acts as raw dataset.

TABLE II: Seed Statements along with Example Uutput

Seed No.	Seed Statement	Example Output
1	FUNC_PROG_START CMD_DECL SCOPE_OFVAR DTYPE	Write a function to creates global integer
2	FUNC_PROG_START CMD_DECL SCOPE_OFVAR DTYPE_NUM equal to RAND_NUM	Create a program to define global integer equal to 20

Continued on next page

TABLE II: Seed Statements along with Example Uutput (Continued)

Seed No.	Seed Statement	Example Output
3	FUNC_PROG_START CMD_DECL SCOPE_OFVAR DTYPE_NUM VARI=RAND_NUM	Write a program for create number k=18
4	FUNC_PROG_START CMD_DECL SCOPE_OFVAR DTYPE VARI	Create a function to declares integer int0
5	FUNC_PROG_START CMD_DECL SCOPE_OFVAR DTYPE_NUM to RAND_NUM	Create program for create local nums to 59
6	FUNC_PROG_START CMD_DECL SCOPE_OFVAR DTYPE equal VARI to RAND_NUM	set local variable equal j to 77
7	FUNC_PROG_START CMD_DECL SCOPE_OFVAR DTYPE_CHAR = 'hi'	Write function to set characters = 'hi'
8	FUNC_PROG_START CMD_DECL SCOPE_OFVAR DTYPE_CHAR to 'hi'	set characters to 'hi'
9	FUNC_PROG_START CMD_DECL SCOPE_OFVAR VARI = 'hi'	Write a function to creates global lists1 = 'hi'
10	FUNC_PROG_START CMD_DECL SCOPE_OFVAR VARI to 'hi'	Write a function for create global txt_1 to 'hi'
11	FUNC_PROG_START CMD_DECL SCOPE_OFVAR DTYPE_CHAR VARI = 'please enter value'	Write a program for set local char char_0 = 'please enter value'
12	FUNC_PROG_START CMD_DECL SCOPE_OFVAR DTYPE_CHAR VARI to 'please enter value'	create character j to 'please enter value'
13	FUNC_PROG_START CMD_PRNT VARI	Write function to show strs0
14	FUNC_PROG_START CMD_PRNT 'please select value'	show 'please select value'
15	FUNC_PROG_START CMD_PRNT 'please select value' + VARI	write 'please select value' + str1
16	FUNC_PROG_START CMD_PRNT VARI + i	display c + i
17	FUNC_PROG_START CMD_PRNT 'please select value' + i	Create a program to shows 'please select value' + i
18	FUNC_PROG_START CMD_INPUT VARI	Create function for input string_1
19	FUNC_PROG_START CMD_INPUT 'please enter name'	Create function for insert 'please enter name'
20	FUNC_PROG_START add VARI + VARI	Create a function to add strings0 + strings_0
21	SPL_FUNC VARI to SPL_CLASS	print strs_2 to screen
22	Write a FUNC_PROG FUNC_NAME to perform OPER with VARI, DTYPE_NUM VARI = RAND_NUM	Write a program prog_div to perform add with i, floats j = 47
23	FUNC_PROG_START OPER SCOPE_OFVAR DTYPE_NUM VARI and VARI	Create a function for mod local number c and address

Table III depicts time taken for each seed statement to generate their respective statements, their individual time and the amount of raw statements it generates. The full seed statements can be seen in Table II. It can observe that statements that have large number of nutrients takes more time to execute and generate more statements compared to ones that have small number of nutrients.

B. Varying NLP Hyper-parameters

While training this NLP models, there were various hyper-parameters like learning rate, batch size, dropout rate, total epochs, etc. Changing those hyper-parameters resulted in different loss and execution time. In Table IV, shows various effects of changing those hyper-parameters.

The first column is kept as the base for benchmark as this is giving the best possible loss value along with relatively fast training time. The other columns shows gradual changes in various parameters, the changed parameters are shown as **bold** while the unchanged parameters (compared to first column) are normal.

TABLE III. TIME TAKEN TO GENERATE INPUT DATA STATEMENTS FROM SEED STATEMENT ALONG WITH THE AMOUNT EACH STATEMENT GENERATES

Seed No.	Individual Time	Cumulative Time	No. of Statements Generated
1	0.049s	0.049s	15288
2	0.022s	0.073s	7176
3	0.438s	0.512s	174096
4	0.383s	0.900s	271440
5	0.031s	0.931s	7176
6	0.721s	1.653s	271440
7	0.014s	1.668s	4056
8	0.015s	1.684s	4056
9	0.077s	1.762s	91728
10	0.078s	1.840s	91728
11	0.150s	2.006s	136656
12	0.172s	2.179s	136656
13	0.015s	2.195s	30576
14	0.014s	2.210s	104
15	0.031s	2.241s	30576
16	0.027s	2.269s	30576
17	0.000s	2.270s	104
18	0.009s	2.279s	19110
19	0.000s	2.279s	65
20	0.000s	2.279s	3822
21	0.000s	2.279s	1470
22	18.621s	20.901s	8195904
23	0.552s	21.700s	369954

TABLE IV. VARIOUS HYPER-PARAMETERS CONFIGURATION AND THEIR EFFECTS BASED ON EPOCHS

	Learn Rate: 0.001 Batch Size: 1000 Dropout Rate: 0.0	Learn Rate: 0.001 Batch Size: 1000 **Dropout Rate: 0.1**	Learn Rate: 0.001 **Batch Size: 5000** Dropout Rate: 0.0	**Learn Rate: 0.005** Batch Size: 1000 Dropout Rate: 0.0	**Learn Rate: 0.005 Batch Size: 5000 Dropout Rate: 0.1**	Learn Rate: 0.001 **Batch Size: 5000 Dropout Rate: 0.1**
Epoch 1/5	Losses: 143006.98 Time: 90.46s	Losses: 173171.71 Time: 102.71s	Losses: 576695.25 Time: 85.97s	Losses: 52150.45 Time: 91.47s	Losses: 278816.12 Time: 101.01s	Losses: 611403.16 Time: 99.14s
Epoch 2/5	Losses: 14.13 Time: 178.95s	Losses: 14.34 Time: 206.14s	Losses: 89620.88 Time: 172.35s	Losses: 95.10 Time: 188.07s	Losses: 435.87 Time: 201.79s	Losses: 200406.14 Time: 199.16s
Epoch 3/5	Losses: 7.49 Time: 269.70s	Losses: 0.94 Time: 313.47s	Losses: 1655.89 Time: 259.04s	Losses: 219.68 Time: 298.05s	Losses: 81.57 Time: 306.28s	Losses: 9531.23 Time: 300.40s
Epoch 4/5	Losses: 0.00014 Time: 363.22s	Losses: 2.53 Time: 422.21s	Losses: 11.83 Time: 348.93s	Losses: 54.44 Time: 424.99s	Losses: 87.22 Time: 414.40s	Losses: 481.82 Time: 405.47s
Epoch 5/5	Losses: 1.6228e-06 Time: 457.60s	Losses: 0.0014 Time: 531.43s	Losses: 1.77 Time: 440.41s	Losses: 2.00 Time: 563.29s	Losses: 26.73 Time: 524.79s	Losses: 7.43 Time: 511.42s

C. Input Statement

We used the below 2 statements to demonstrate the output of our system during various stages of the system. These are the example of inputs that the user might use during its production phase.

Statement 1:

define variable text1 = 'please enter value'

Statement 2:

create a function to add a and local number num_1 = 15 and b

D. NER System Output

Fig. 6 and 7 are the outputs given by the NER model. Each word (token) is bundled with original word given by the user and tag assigned by the NER model. In the left is the word/token and in the right is tag assigned.

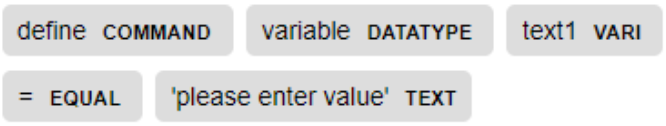

Fig. 6. NER Model Output of Statement 1.

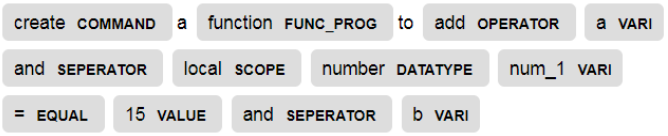

Fig. 7. NER Model Output of Statement 2.

E. Pseudocode Output

Pseudocode file generated by our system is shown below in listings 1 and 2.

```
<?xml version="1.0" ?>
<root>
   <statement value="define variable text1 = 'please
        enter value'"/>
   <program type="program" name="define_variable">
      <variables>
          <variable name="text1" datatype="variable"
               value="'please enter value'"/>
      </variables>
   </program>
</root>
```

Listing 1: Pseudocode output of statement 1

```
<?xml version="1.0" ?>
<root>
   <statement value="create a function to add a and
        local number
     num_1 = 15 and b"/>
   <program type="function" name="create_function">
      <variables>
          <variable name="a" datatype="number"/>
          <variable name="num_1" scope="local"
              datatype="number" value="15"/>
          <variable name="b" datatype="number"/>
      </variables>
      <add variable1="a" variable2="num_1" variable3
          ="b"/>
   </program>
</root>
```

Listing 2: Pseudocode output of statement 2

F. Final Source-code

The source-code is the final output generated. This is the only output visible to the user. Listings 3 and 4 shows the output code.

```
def main():
   variable text1 = 'please enter value'

if __name__ == "__main__":
   main()
```

Listing 3: Source-code output of statement 1

```
def create_function():
   number a
   number num_1 = 15
   number b

   answer = a + num_1 + b

create_function()
```

Listing 4: Source-code output of statement 2

The final results as obtained in our program that is made using python can be seen in Fig. 8, 9, 10 and 11.

XI. Future Work

One of the ways we can further improve on this project is to create support for more programming languages. Currently we have only implemented support for python and C. We can also add support for new common functions like date, time, string operations, etc. Loops like "for", "while" and "do while" are also left out due to its complexity and time constraints which can be expanded later.

Our approach serves as the basic idea that allows the development of systems that are more complex in terms of keyword detection and contains more functionalities like loops and branching by adding additional NLP elements like dependency parsing and Semantic parsing.

XII. Conclusion

This proposal shows that a language-independent solution is a feasible alternate for writing source-code without having full knowledge about a programming language.

Using XML based pseudo code as an intermediate step makes this method as programming language-independent which solves the major drawbacks in existing research that comes with a rigid commitment to only one programming language. The next step, which is converting XML based pseudo code into language-dependent source-code is dependent on premade language format which is modifiable by anyone, making this approach module based approach. If a person wishes to convert the natural language into some other programming language, they simply need to duplicate the premade language format and fill it with their desired programming language keywords.

Another challenge faced in this area which is program based NLP is that, there is not many datasets available in this particular sub-field which severely limits the research capabilities and keeps this sub-field from growing forward. Although not ideal, our dataset generation system provides an automated approach to create thousands of data that can be used in an AI system as a training dataset.

One way this system can be used is with ticket based programming, where the programmers get their tasks in the form of tickets in their mail. The system can be used as a suggestion system where the mail is analyzed and a suggested solution is provided to the programmer. Another use is to pair this system with voice recognition and let the programmer write simple source-code only through speech making their hands free for writing other more complex code.

References

[1] S. Oualline, *Practical c Programming, Third Edition*, 3rd ed. O'Reilly Media, Inc., 1997.

[2] C. W. Thompson and K. M. Ross, "Natural-language interface generating system," in U. S. Patent. U.S., 1987.

[3] S.-Y. Park, J. Byun, H.-C. Rim, D.-G. Lee, and H. Lim, "Natural language-based user interface for mobile devices with limited resources," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 4, pp. 2086–2092, 2010.

[4] E. U. Reshma and P. C. Remya, "A review of different approaches in natural language interfaces to databases," in *2017 International Conference on Intelligent Sustainable Systems (ICISS)*, 2017, pp. 801–804.

[5] P. Gupta, A. Goswami, S. Koul, and K. Sartape, "Iqs-intelligent querying system using natural language processing," in *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, vol. 2, 2017, pp. 410–413.

[6] A. Das and R. C. Balabantaray, "Mynlidb: A natural language interface to database," in *2019 International Conference on Information Technology (ICIT)*, 2019, pp. 234–238.

[7] M. Uma, V. Sneha, G. Sneha, J. Bhuvana, and B. Bharathi, "Formation of sql from natural language query using nlp," in *2019 International Conference on Computational Intelligence in Data Science (ICCIDS)*, 2019, pp. 1–5.

[8] A. Kate, S. Kamble, A. Bodkhe, and M. Joshi, "Conversion of natural language query to sql query," in *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, 2018, pp. 488–491.

[9] L. Tang, X. Mao, and Z. Zhang, "Language to code with open source software," in *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, 2019, pp. 561–564.

[10] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, 2018, pp. 933–944.

[11] T. Ahmad and N. Ahmad, "A simple guide to implement data retrieval through natural language database query interface (nldq)," in *2019 8th International Conference System Modeling and Advancement in Research Trends (SMART)*, 2019, pp. 37–41.

[12] A. Mohite and V. Bhojane, "Natural language interface to database using modified co-occurrence matrix technique," *2015 International Conference on Pervasive Computing (ICPC)*, pp. 1–4, 2015.

[13] S. S. Badhya, A. Prasad, S. Rohan, Y. S. Yashwanth, N. Deepamala, and G. Shobha, "Natural language to structured query language using elasticsearch for descriptive columns," in *2019 4th International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS)*, vol. 4, 2019, pp. 1–5.

[14] T. Dirgahayu, S. N. Huda, Z. Zukhri, and C. I. Ratnasari, "Automatic translation from pseudocode to source code: A conceptual-metamodel approach," in *2017 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom)*, 2017, pp. 122–128.

[15] L. Haowen, L. Wei, and L. Yin, "An xml-based pseudo-code online editing and conversion system," in *IEEE Conference Anthology*, 2013, pp. 1–5.

[16] A. T. Imam and A. J. Alnsour, "The use of natural language processing approach for converting pseudo code to c# code," *Journal of Intelligent Systems*, vol. 29, no. 1, pp. 1388–1407, 2020. [Online]. Available: https://doi.org/10.1515/jisys-2018-0291

[17] M. Balog, A. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow, "Deepcoder: Learning to write programs," 11 2016.

[18] M. Hasan, K. S. Mehrab, W. U. Ahmad, and R. Shahriyar, "Text2app: A framework for creating android apps from text descriptions," *ArXiv*, vol. abs/2104.08301, 2021.

[19] A. Alhefdhi, H. K. Dam, H. Hata, and A. Ghose, "Generating pseudo-code from source code using deep learning," in *2018 25th Australasian Software Engineering Conference (ASWEC)*, 2018, pp. 21–25.

[20] F. F. Xu, Z. Jiang, P. Yin, B. Vasilescu, and G. Neubig, "Incorporating

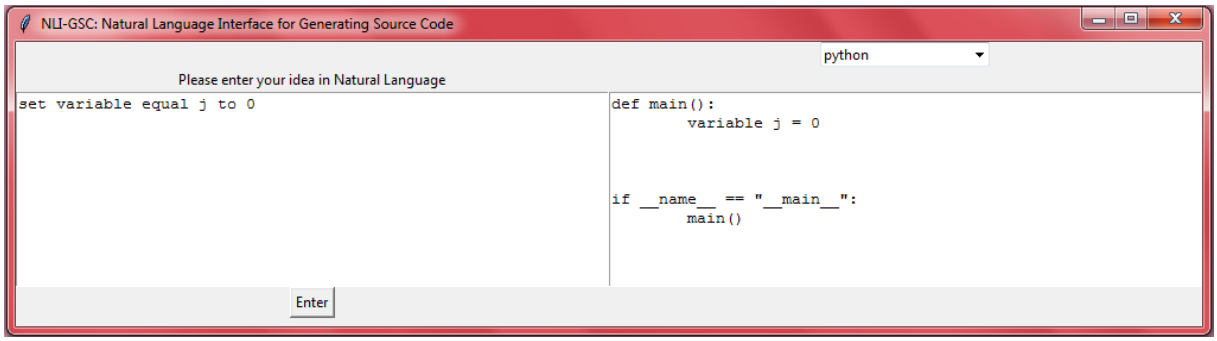

Fig. 8. Screenshot of our Program 1.

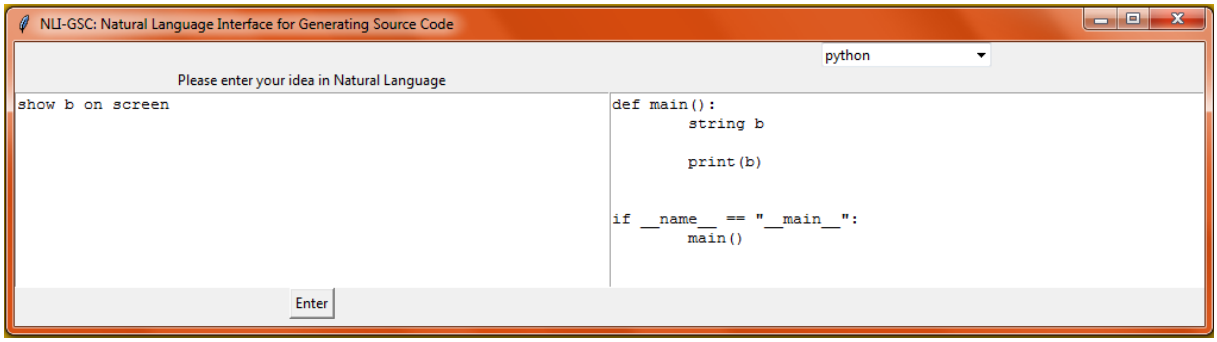

Fig. 9. Screenshot of our Program 2.

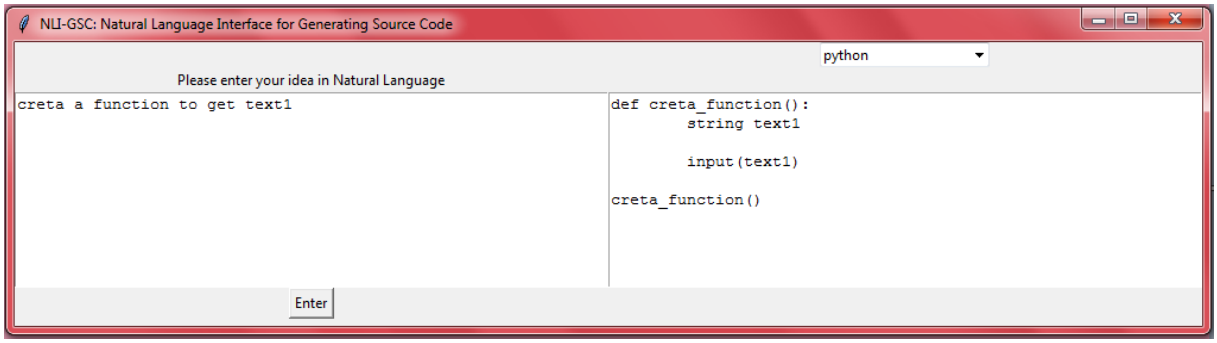

Fig. 10. Screenshot of our Program 3.

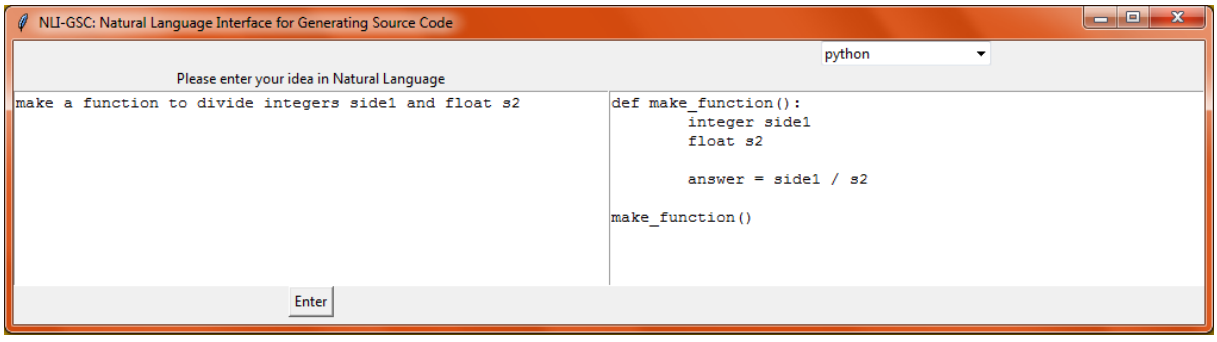

Fig. 11. Screenshot of our Program 4.

external knowledge through pre-training for natural language to code generation," in *ACL*, 2020.

[21] F. F. Xu, B. Vasilescu, and G. Neubig, "In-ide code generation from natural language: Promise and challenges," *ArXiv*, vol. abs/2101.11149, 2021.

[22] M. Bosnjak, T. Rocktäschel, J. Naradowsky, and S. Riedel, "Programming with a differentiable forth interpreter," in *ICML*, 2017, pp. 547–556.

[Online]. Available: http://proceedings.mlr.press/v70/bosnjak17a.html
[23] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997.
[24] W. Weaver, "Translation," in *Machine Translation of Languages*, W. N. Locke and A. D. Boothe, Eds. Cambridge, MA: MIT Press, 1949/1955, pp. 15–23, reprinted from a memorandum written by Weaver in 1949.
[25] N. Chomsky, "Three models for the description of language," *IRE Transactions on Information Theory*, vol. 2, no. 3, pp. 113–124, 1956.
[26] J. B. Lovins, "Development of a stemming algorithm," *Mech. Transl. Comput. Linguistics*, vol. 11, pp. 22–31, 1968.
[27] M. Porter, "An algorithm for suffix stripping," *Program*, vol. 40, pp. 211–218, 1980.
[28] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.
[29] G. Forney, "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.
[30] D. Bikel, S. Miller, R. Schwartz, and R. Weischedel, "Nymble: a high-performance learning name-finder," 04 1998.
[31] F. Lundh, "An introduction to tkinter," *URL: www. pythonware. com/library/tkinter/introduction/index. htm*, 1999.
[32] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.
[33] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," 2017, to appear.
[34] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Association for Computational Linguistics (ACL) System Demonstrations*, 2014, pp. 55–60. [Online]. Available: http://www.aclweb.org/anthology/P/P14/P14-5010
[35] P. Silva, C. Gonçalves, C. Godinho, N. Antunes, and M. Curado, "Using natural language processing to detect privacy violations in online contracts," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, ser. SAC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1305–1307. [Online]. Available: https://doi.org/10.1145/3341105.3375774
[36] S. Jugran, A. Kumar, B. S. Tyagi, and V. Anand, "Extractive automatic text summarization using spacy in python & nlp," in *2021 International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, 2021, pp. 582–585.
[37] A. V. Aho, "Algorithms for finding patterns in strings, handbook of theoretical computer science (vol. a): algorithms and complexity," 1991.