CodeGenAI: A Creative Approach for Code Generation

Project Description and Design for CodeGenAI

1. Introduction

Programming is considered as a creative process. Large systems are built with the programming of various system components and complex logic. There are 800+ programming languages and among all of them, almost 100+ languages are used currently[Ref]. The question then becomes that Is programming language barrier, degrade creativity of programmer for building overall systems? The eminent purpose building ***CodeGenAI as a creativity support tool that generates code in selected language from natural language text to enhance programmer's creativity***. Also, to get insight for above statements, we have conducted initial experiments on 15 Georgia Tech Computer Science students, who are recently introduced to at least one new programming language.

2. Motivation

One of the recommendable property of creative system should be, Agent has to keep evolving its structure and keep colonizing its base program with the meta-reasoning mechanism. Continuous learning gives the agent ability to adapt to the environment and re-develop itself with variable inputs. The first milestone for that is to make creativity support tool that will generate code from natural language and can be work as an autonomous system without language dependency. The purpose is to enhance co-creativity with the machine and not to work autonomously on code generation.

3. Problem Statement

CodeGenAI takes natural language text as input and generates code in one of the selected languages as output.

Talking about natural language input, itself describe ambiguity in input and very large possibility over input space. However, the complexity of input is intermediate level. It should not as simple as it will increase the actual length of the code. It should be as concise as CodeGenAI can extract keywords in patterns.

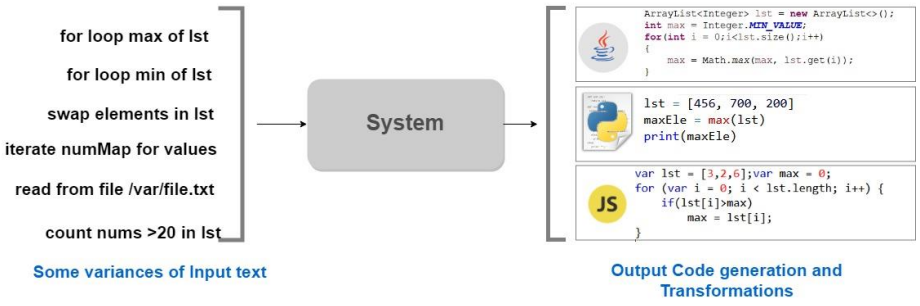

Figure 1: Abstract Design in terms of Input and Output (design tool draw.io)

Figure 1, shows exact level of abstraction for input text. The agent will take text language instruction concisely and output relevant code in the selected language (Java, Python, JavaScript, etc). The text will not be as abstract text as it increases the length of the original code and It would not as specific as the code itself. The user will also provide output language. (In which language

System should give output). Same input sentence can transform code snippets in a different language.

4. Model Selection and Decomposition of Tasks.

For creating CodeGenAI we had experiment different methodology and techniques for text to code translation. Following describes detail methodologies and outcomes for the code generation purpose.

First, we have tried to make program dependency graph from the input text and assign similar structure graph with code snippets. To generate code from the input text, it will check for graph isomorphism with respective graph and return code as output. However, considerable thought is graph isomorphism for the large graph would take more time in comparison. Also, for input combinations of text can result in different graphs for the same context of the sentence.

Second, we tried to use latent semantic indexing method to distribute probability of next occurring word in the input text. This is helpful to find a pattern of words in a sentence and accordingly can change the code in correct sequential manner. Nonetheless, it lacks storage structure for code and can lead to the correct result but wrong code generation.

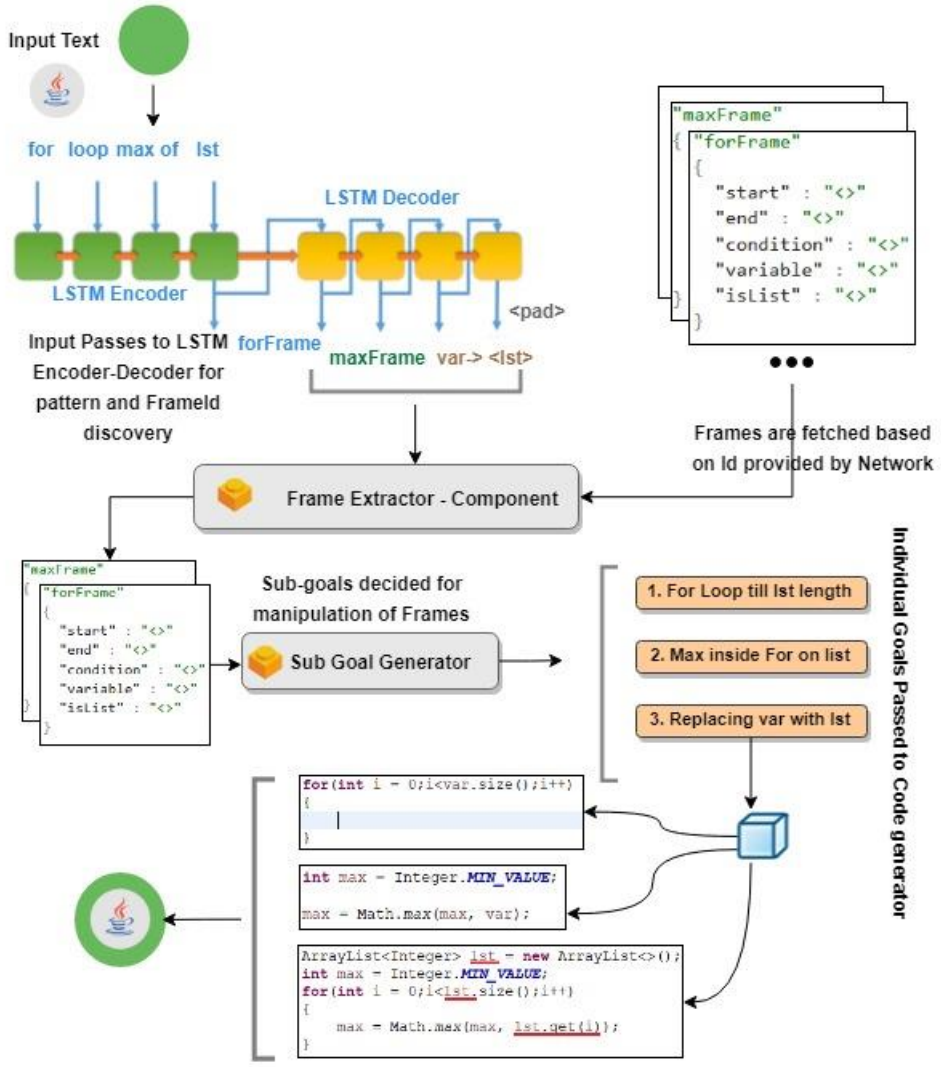

Figure 2: Detailed Task Decompositions with Example (design tool draw.io)

These failures lead us to the idea to use neural machine translation in the analogy of programming language. Can we use LSTM to predict pattern as well as define storage structures for input text? thinking of answer inspired for the following design on component perspective, 1. Input text passed to Neural machine translation engine for finding the id of the frame(s) related to input text. 2. Respective Frame is fetched and passed as input for Goal Generator Component. 3. It will generate sub-goals for the input sentence. 4. The code will be generated and merged for each sub goals and output in one of the selected language.

1. From code editor, natural language text passed as input to LSTM encoder, and as result, we will get respective frameIDs in sequence from decoder layer. The component uses neural machine translation on the programming language and maps it to output for text. This will give us information about frames to use and sequence of that frame.

2. Frame sequence fetched from decoder will be passed futher. For example, **forFrame** and **maxFrame** fetched from dynamic frame factory and passed to sub goal generator for creating sub-goals to fill that frame.

3. Goal Generator plays the crucial role in filling slots and generating individual code. It will take an input string and frames as input and decide three sub goals (1) Generating for loop for a list (2) Find Max and (3) Replace generic variables with **lst.**

4. Code Generator will generate code for each sub tasks and merge this code. Methodology: Each frame will contain language wise code snippet implementation for that frame. This gives code generator potential to replace place holders and output accurate code in given language.

5. Learning with Feedback Loop

One reasonable observation from design leads to question that how much data do we need to actually train the LSTM model. Combination of input text may lead to the infinite trivial combination for the code. To overcome this challenge, we decided to introduce feedback loop mechanism.

When we write code in the editor with comments, the agent will take this code as input, convert it to the dynamic frame structure (if the frame is available already, use existing frame) and map input comment to frame IDs. This will lead us to one training instance. Also, by permutations of input comment text, we can generate more possible combinations and train model.

We have planning to execute learning by feedback loop before final deliverable of the project.

5.1. User Interface

Figure 1 shows the main user interface for CodeGenAI: a GUI text editor which shows some default code and instructions when initially loaded. The user can select one of the supported languages from the selection box on the top right. At this time, CodeGenAI supports 3 languages: Java, JavaScript and Python.

Figure 1: CodeGenAI User Interface

Figure 2 exhibits two main functionalities in CodeGenAI: populating code based on natural language input (by typing a comment as the natural language input and pressing Ctrl+H), and learning code by code selection (by selecting a chunk of code to learn and pressing Ctrl+L). Learning code chunks helps users remember certain snippets that are commonly used, but not yet supported in CodeGenAI.

It is very important for CodeGenAI to generate code without syntax or semantic errors. For code checking in the editor, we have added static code evaluation. This will check for syntax errors and notify the user in the left panel if errors are detected, with appropriate messages provided.

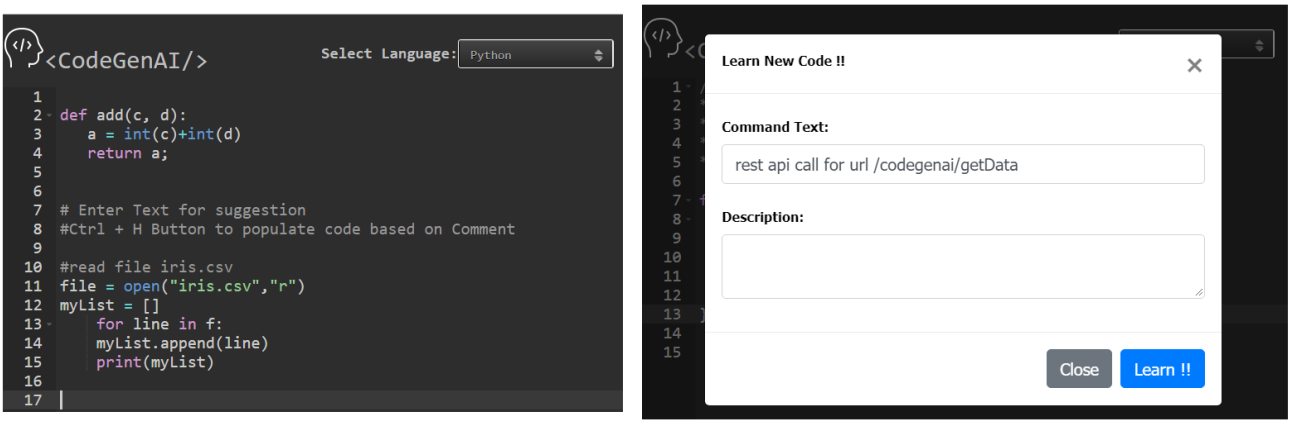

Figure 2: Generating code and learning code

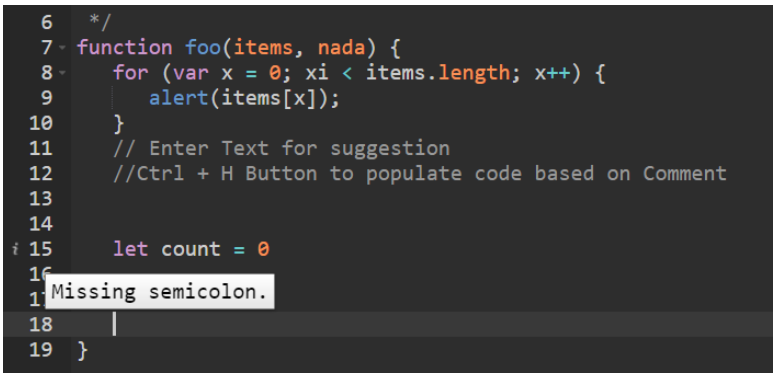

Figure 3: Checking of syntax errors in CodeGenAI

Figure 3 shows an example of code checking in action. In the Java editor, if a semicolon is missing in code, a pop-up message alerts the user to the mistake, so that the user can easily understand and correct it. In our evaluations (discussed in Section 4), 83% of users who saw such messages were able to draw analogies to similar issues in programming languages they had experience in and correct the mistakes highlighted.

6. Value Addition in Computational Creativity

6.1 As Product

As a product, the system is intended to play a role in creativity support, helping novice and experienced programmers alike by helping them focus more on the higher-level abstraction of the program, design, and problem solving than to be bogged down by syntax, language rules, and coding.

Likewise, programmers new to a language will be able to learn the basics of the language quickly because the system significantly reduces the learning curve of learning to program and they can use the extra time gained from not coding to analyze and recognize patterns that may apply to several languages and understand concepts. Novices can either use this system to generate simple programs more quickly or as an educational tool in how that description is interpreted and translated to a program.
Experienced programmers, whether they are building a game, an app, or code for a creative system like DARCI, can use this system to generate code that performs functionalities they require, so they can think at a higher level of abstraction, allowing more time for creative ideas to be generated and tested.

6.2 As Process

There are two main components that uniquely modified in the code generation process. First, Using Neural machine translation in coding language depicts novel idea in terms of the analogy of pattern finding. Using a sequence to sequence encoder, the system is able to encode a natural language input that can be decoded to a tag that acts as a pointer to a code unit in the knowledge base. The system will work if the pattern finding based on the comments or code description can can decipher meaning from complex and varying data. By having a large dataset of code descriptions that can be mapped to many frames, the neural net will be able to pick the correct frame even if the functionality or generated code is very similar.

Second, usage of frames as code generation is another way to accurately generate and merge code. Frames contain the basic structure for a particular functional code unit. Once a frame is selected then the values are filled in and the code is generated.

7. Evaluation

7.1 Evaluation of Co-Creativity

As discussed in class with Professor Goel, after Thursday meeting, we have conducted studies on 15 Georgia Tech students to know creativity evaluation measure. Following graph depicts high there might be possible that answers might tend to high bias towards one perspective.

We are planning to measure co-creativity as a combination of summative and formative evaluation. More detailed description in section 7.3.

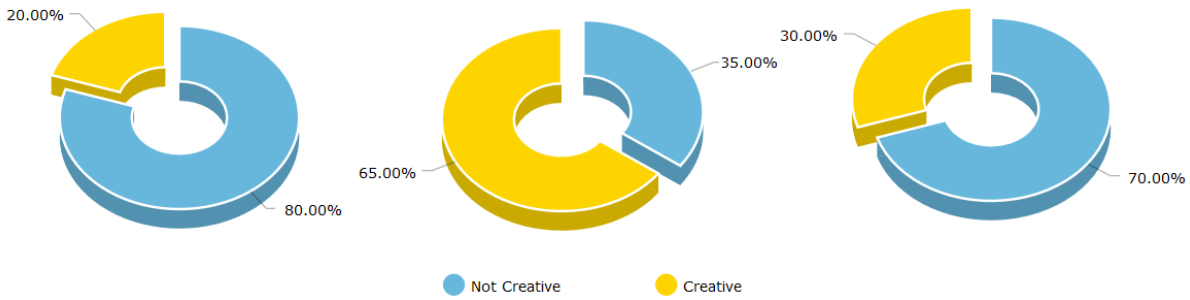

Figure 3: Small Study on Human Bias

7.2 Evaluation of Process

We are planning to evaluate the process of generating code with a formative measure of input text provided. (change syntax of input text should not change output if the meaning of a text is same). We believe that code generation process can be measure in perspective of creativity. We are still in process of conducting user studies for evaluation measures that will effectively evaluate the process.

One way to quantitatively measure the process of code generation is to use Ritchie's measurement of novelty and quality using the functions typ and val respectively. The system lends perfectly to be tested against Ritchie's 14 criteria. There is the inspiring set of existing codebases, then pattern finding and frames that acts as a mapping function from the inspiring set, and initial data values which are the frames and the data structures the user wants the generated code to manipulate. If the generated code captures the meaning of the natural language input and outputs generated code that is different from the inspiring set, but still follows the structure outlined by frames and works then according to Ritchie's criteria the system will have performed creatively and will have satisfied the typicality, achieved by code using structure from frame, and quality, achieved if the generated code performs right operations on inputted data structure.

To measure typicality and quality, a study where users input natural language input and grade the generated code based on typicality and quality. The study participants would need to be a detailed description of the inspiring set and what it means to be typical and have quality in that domain so that their biases will not affect the ratings.

7.3 Experiments

Recent user experiments consist of 4 questions about, basic experience programming and difficulties programmer face during the semester to work on the new programming language. The analysis shows the result that about 93% of the population face difficulties while learning syntax and semantics of the new language. Further, detailed conversation leads that programmer think structure/logic of the program in mind and then search for different component syntax on the search engine.

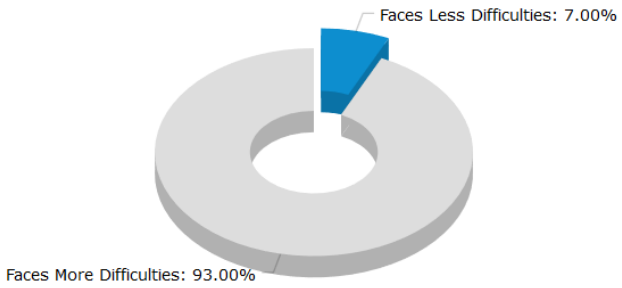

Figure 4: Experimenting difficulties for new language

For further evaluation, we have planned to evaluate two user group for enhancement of creative thinking and productivity. First Group, will write a program for problem statement in the new language with their choice of IDE. The other group can use CodeGenAI as a support tool for programming. We will analyze, quality of code and logic implementation as the measure of creativity.

We will conduct this experiment as soon as we completed prototype design in such a manner that can be published to the user.

References

[1] A. Perez-Poch, N. Olmedo-Torre, F. Sanchez, N. Salan, and D. Lopez, "On the influence of creativity in basic programming learning in a first-year engineering course," *International Journal of Engineering Education,* vol. 32, no. 5, pp. 2302-2309, 2016.

[2] T. Jenkins, "On the difficulty of learning to program," *Proceedings of the 3rd Learning and Teaching Support Network for Information and Computer Science Conference,* pp. 65-71, 2002.

[3] W. C. Hsu and Y. Mimura, "Understanding the secondary digital gap: Learning challenges and performance in college introductory programming courses," in *2017 IEEE 9th International Conference on Engineering Education (ICEED), 9-10 Nov. 2017*, Piscataway, NJ, USA, 2017, pp. 59-64: IEEE.

[4] GitHub. (2017). *GitHub Octoverse 2017 | Highlights from the last 12 months*. Available: https://octoverse.github.com/

[5] M. Resnick *et al.*, "Scratch: programming for all," *Commun. ACM,* vol. 52, no. 11, pp. 60-67, 2009.

[6] R. Clayton, S. Rugaber, and L. Wills, "Dowsing: a tool framework for domain-oriented browsing of software artifacts," in *Proceedings 13th IEEE International Conference on Automated Software Engineering, 13-16 Oct. 1998*, Los Alamitos, CA, USA, 1998, pp. 204-7: IEEE Comput. Soc.

[7] H. Fudaba *et al.*, "Pseudogen: A Tool to Automatically Generate Pseudo-Code from Source Code," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), 9-13 Nov. 2015*, Los Alamitos, CA, USA, 2015, pp. 824-9: IEEE Computer Society.

[8] E. Wong, T. Liu, and L. Tan, "CloCom: Mining existing source code for automatic comment generation," in *22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015, March 2, 2015 - March 6, 2015*, Montreal, QC, Canada, 2015, pp. 380-389: Institute of Electrical and Electronics Engineers Inc.

[9] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *28th Annual Conference on Neural Information Processing Systems 2014, NIPS 2014, December 8, 2014 - December 13, 2014*, Montreal, QC, Canada, 2014, vol. 4, pp. 3104-3112: Neural information processing systems foundation.

[10] (2015). *Understanding LSTM Networks -- colah's blog*. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[11] C. Holmes and A. Evans, "A Review of Frame Technology," York University, UK2003.