Intelligent Code Editor with Code Suggestion and Code Correction

Design Document

Team 29
Iowa State University (Ali Jannesari)
Ali Jannesari
Jacob Puetz, Ben Gonner, Cory Smith, Evan Christensen, Jordan Silvers
sdmay21-29@iastate.edu
Team Website
https://sdmay21-29.sd.ece.iastate.edu/

Revised: 10/25/20

Executive Summary

Development Standards & Practices Used

We used standard industry practices as taught in our previous classes while working on the Intelligent Code Editor. We are planning on using Test-Driven Development as well for the project as it has been highly praised by professors throughout the Software Engineering and Computer Science departments. This will allow us to test the quality of the Intelligent Code Editor based on the number of bugs we discover from our process. We are also planning on implementing the design thinking process to allow us to be able to empathize with the users and clients on the Intelligent Code Editor.

Summary of Requirements

- Ability for user to input code and natural language into an editor
- Ability to store the natural language and expected code translation
- Ability for plugin to classify and translate natural language to code
- Ability for user to translate the natural language to code
- Ability for code to be executed once translated
- Ability for multiple natural language statements to be translated to code at once
- Fix any bugs that exist from previous versions

Applicable Courses from Iowa State University Curriculum

- COMS 227 (Introduction to Object-Oriented Programming)
- COMS 228 (Introduction to Data Structures)
- COMS 309 (Software Development Practices)
- COMS 230 (Discretes CMP Structures)
- COMS 311 (Algorithm Design and Analysis)
- SE 329 (Software Project Management)
- SE 339 (Software Architecture and Design)
- SE 362 (Object-Oriented Analysis and Design)

New Skills/Knowledge acquired that was not taught in courses

List all new skills/knowledge that your team acquired which was not part of your Iowa State curriculum in order to complete this project.

- IDE plugin Development
- Graph Neural Networks
- Semi-Supervised Learning for Neural Machine Translation
- Unsupervised Machine Translation
- Natural Language Processing
- Data Formatting and Cleansing

Table of Contents

List of figures/tables/symbols/definitions (This should be the similar to the project plan)

1 Introduction

1.1 Acknowledgement

This project was started by last year's senior design team sdmay 20-46.

1.2 Problem and Project Statement

Our project is an extension for a code editor that takes natural language statements and translates them into functional code. We plan to achieve this by implementing a graph neural network to classify words and phrases. This extension could change the way we approach programming. It offers a new way to think about software problems and a more natural transition from traditional solutions to syntactically correct solutions. We will expand upon this extension by implementing code corrections and suggestions as well. This will further improve the user experience for users new to programming and help them learn.

1.3 Operational Environment

The end result of our project is a plugin for the IntelliJ IDEA Java IDE. As such, it will be able to run on any computer capable of running the IDE. Additionally, the computer will need to have Python and OpenNMT-py installed, and be actively running the OpenNMT-py server for the plugin to be operational.

1.4 Requirements

- To create a plugin to Intellij that is able to convert natural language to code.
- Fix any bugs that exist from previous developers.
- Refine and add code corrections and suggestions to help developers make better programs.
- Users should be able to input both code and natural language into their editor.
- The plugin should store both natural language and its expected translation.
- The plugin should be able to translate natural language to equivalent code.
- The plugin should allow users to convert their natural language statement to code after the translation is complete.
- The code translation should run as-is, without modification by the user.
- The plugin should be able to translate multiple lines of natural language at once.
- The project will be entirely digital.
- All work on the project will be done remotely.

1.5 Intended Users and Uses

Our plugin is designed for inexperienced Java developers who are still learning how to convert their ideas into real code. With this in mind, our main users will be students in introductory programming classes or individuals who are learning to program on their own. These users will use

the plugin to convert their natural language statements into code and to find common errors in their code to make their programs run the way they want.

1.6 Assumptions and Limitations

Assumptions:

- The end user has an internet connection when they are programming.
- The end user will be able to pick the best code suggestions that fit their purpose.

Limitations:

- The final product will not work without an internet connection as it needs to access a code translation server.
- Currently only works with Intellij for functionality implementation.

1.7 Expected End Product and Deliverables

We will improve upon the current system for translating natural language phrases into syntactically correct code snippets. November 2020.

We will write a document describing instructions for users to install the extension and other initial set-up steps. April 2021.

We will expand upon the extension to provide suggested corrections for common mistakes, and auto complete code snippets. April 2021.

2 Project Plan

2.1 Task Decomposition

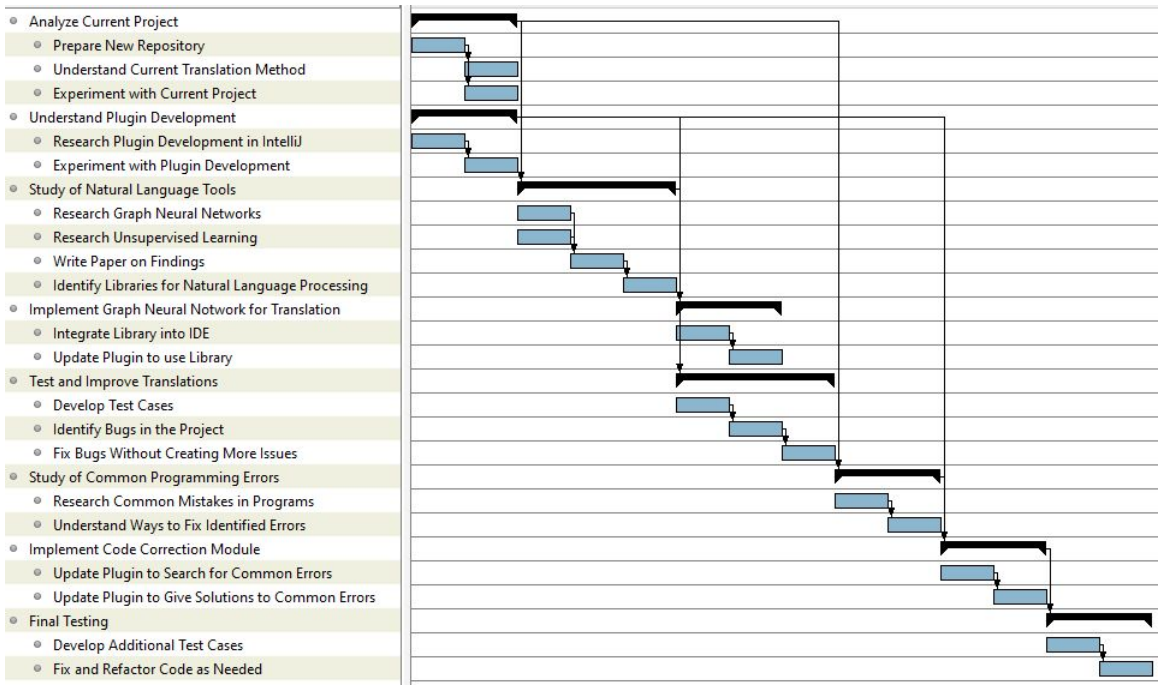

Figure 1: Gantt Chart

Analyze Current Project

- Prepare new repository: Move code from the old repository to the new repository.
- Understand Current Translation Method: Understand how the version one code translates natural language.
- Experiment with Current Project: Experiment with the version one code on our local machines.

Understand Plugin Development

- Research Plugin Development in IntelliJ: Research how plugin development works in our current IDE (IntelliJ).
- Experiment with Plugin Development: Experiment with plugin development on our local machines.

Study of Natural Language Tools

- Research Graph Neural Networks: Research how graph neural networks can be used in natural language processing.
- Research Unsupervised Learning: Research how unsupervised learning can be applied in the project.

- Write Paper on Findings: Write a paper to deliver to the client on how graph neural networks and unsupervised learning can be used.
- Identify Libraries for Natural Language Processing: Identify libraries that could be used within our project to translate natural language into code.

Implement Graph Neural Network for Translation

- Integrate Library into IDE: Download our chosen library and add it into our project.
- Update Plugin to Use Library: Update the plugin to make it use the graph neural network library instead of the old system.

Test and Improve Translations

- Develop Test Cases: Develop some test cases to ensure that the project meets the client's requirements.
- Identify Bugs in the Project: Identify bugs in the current project based on test cases.
- Fix Bugs Without Creating More Issues: Fix the identified bugs so that the project meets its requirements without introducing new errors.

Study of Common Programming Errors

- Research Common Mistakes in Programs: Research common mistakes that developers make when working on programs.
- Understand Ways to Fix Identified Errors: Determine the best ways that the identified mistakes can be resolved.

Implement Code Correction Module

- Update Plugin to Search for Common Code Errors: The plugin should be capable of finding common code errors as well as natural language.
- Update Plugin to Give Solutions to Common Errors: Expand the plugin's capabilities to give suggestions for common code errors as well as natural language translations.

Final Testing

- Implement Tests: Create tests for many use cases, including corner cases for maximum coverage.
- Fix and Refactor the Code as Needed: As tests fail to pass, make any necessary changes to the code base to fix the problem.

2.2 Risks And Risk Management/Mitigation

Analyze Current Project

- If the plugin's code is not understood by the team future development could be slowed. 10%
- error getting the team members machines to run the code might occur. 90%

    - Mitigation plan: discuss issues running software with the client for insight.

Understand Plugin Development

- If the plugin development resources are not understood future development could be slowed. 5%

Study of Natural Language Tools

- Struggling to comprehend graph neural networks will slow or halt progress. 20%

Implement graph neural network for code translation

- An improperly trained neural network will have a low accuracy rate. 14%
- Build errors might occur with the implementation of the module. 30%

Test and Improve Translations

- Fixing a bug has a chance to reveal new bugs. 40%

Study of Common Programming Errors

- People alway make errors and there will probably be a large number of different code errors people make. 15%

Implement Code Correction Module

- Build errors might occur with the implementation of the module. 30%

Final Testing

- Tests are not implemented correctly and bugs that exist are not found through testing. 10%

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

- The extension shall translate natural language into syntactically correct code snippets with 90% accuracy.
- The extension shall translate natural language phrases at a rate of 100 phrases per second.
- The extension shall provide auto-completion suggestions within 10 ms of the last keystroke.
- The extension shall identify code errors with 80% accuracy.

2.4 Project Timeline/Schedule

The first step of the project was the analysis phase. As shown in Figure 3, this phase consists of three parts: prepare, analyze, and experiment. We first move the old code into the new repository, then we analyze the code to gain a better understanding of it, and lastly we experiment with the code.

Secondly, the team started to research IntelliJ plugin development to get an understanding of how plugins are made and how to work on the existing project. The team also might experiment with plugin creation and implementation during the phase.

Third, we looked into Natural language tools. This phase includes researching graph neural networks along with both semi-supervised and unsupervised learning and identifying the current libraries to use. We will then write and present a paper for our client based on our findings.

After looking into natural language tools the team will be implementing a graph neural network to quickly and accurately translate the natural language into code.

We then are going to create tests for the previous implementation. We will develop specific test cases to cover as much area as we can to find bugs in the code. We will then try to fix any bugs that we find without compromising the existing code.

We will develop new features into version 2 of the code. New features added will include identification of common programming errors and suggested fixes for errors found.

Finally, we will add in test cases for the version 2 of the code. This will be a similar process as our test cases before but will be geared more towards the version 2 of the code. We will then identify and fix any bugs we find for version 2. We will also refactor the code as needed so other groups down the line do not have a problem identifying the parts of the program and can work more easily with it after we are gone.

2.5 Project Tracking Procedures

For our project tracking we are using Git, Trello, and Discord. Git allows us to share the code between each other and our client. Trello allows us to keep track of tasks that are planned, in progress, and completed. The Trello is managed/maintained alongside the client. Discord is used as a way for the team to communicate with each other and share helpful information and links.

2.6 Personnel Effort Requirements

Task	Number of Hours
ANALYZE	**11 hours**
Move Old Code to New Repository	1 hour
Understand Version 1 Code	5 hours
Experiment with Version 1 Code	5 hours
RESEARCH	**35 hours**

Research IDE Plug-in Development	5 hours
Research Semi-Supervised Learning	10 hours
Research Unsupervised Learning	10 hours
Research Natural Language Processing	10 hours
VERSION 1 CODE	**70 hours**
Bug Fixes from Version 1 Code	20 hours
Improvements on Natural Learning from Version 1 Code	50 hours
VERSION 2 CODE	**85 hours**
Research Common Types of Erroneous Code	10 hours
Implement Code Correction Model	75 hours
TESTING	**100 hours**
Write Unit and Integration Tests	25 hours
Run Usability Tests	25 hours
Fix and Refactor Code	50 hours

2.7 Other Resource Requirements

Given that our project is a plug-in for Intelli-j, we currently do not have any other needs for physical parts or materials. We are also not expecting to need any physical parts as our project is solely based on the plug-in and the integrated development environment. Our only other dedicated requirement outside of physical is a dedicated GPU server to train our classification model on the dataset.

2.8 Financial Requirements

There are no financial resources required as everything we need to use is free/given to us. We are also not expecting any surprise finances to appear throughout the course of the project.

3 Design

3.1 Previous Work And Literature

Literature Review

Graph Neural Networks: A Review of Methods and Applications from Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, Maosong Sun

Semi-Supervised Learning for Neural Machine Translation from Yong Cheng, Wei Xu, Zhongjun He, Wei He, Hua Wu, Maosong Sun and Yang Liu

UNSUPERVISED MACHINE TRANSLATION USING MONOLINGUAL CORPORA ONLY from Guillaume Lample , Alexis Conneau, Ludovic Denoyer, Marc'Aurelio Ranzato

Learning to Represent Programs with Graphs from Miltiadis Allamanis, Marc Brockschmidt, Mahmoud Khademi

These papers were graciously provided to us by both our client and advisor to give us an introduction to Neural Networks as we had previously told both parties that none of us had any prior knowledge of the topic.

Similar Products

Google Translate

- While not directly related to our project, Google Translate does something similar to ours with taking one language and changing it to another language.

Other Neural Projects

- As we have seen in other papers on the subject, other projects out there exist with a similar task as ours. The difference between ours and theirs is that ours is basically open source. The other projects that have been talked about in other academic papers give no insight as to what is being done in the project and how it is being applied. There is also no code provided from these other projects so it is hard to tell if they are actually completed or more theory based.

3.2 Design Thinking

Detail any design thinking driven design "define" aspects that shape your design. Enumerate some of the other design choices that came up in your design thinking "ideate" phase.

Our project's problem is well defined because we are expanding on the work of a past senior design group. We need to design a system that takes as input natural languages phrases, and produces code snippets translated from the phrases. The client wants the system to be implemented with a graph neural network to attempt to increase the accuracy of the system and expand on new uses for graph neural networks.

While looking for libraries and resources for graph neural networks, we found Spektral as well as a library by Microsoft and another independent library. After presenting these options to our client, we were steered toward Spektral.

We need a graph neural network for our plug-in. We have looked into some different kinds of graph neural networks - graph convolutional neural networks and gated graph neural networks.

3.3 Proposed Design

Include any/all possible methods of approach to solving the problem:

- *Discuss what you have done so far – what have you tried/implemented/tested?*
- *Some discussion of how this design satisfies the **functional and non-functional requirements** of the project.*
- *If any **standards** are relevant to your project (e.g. IEEE standards, NIST standards) discuss the applicability of those standards here*
- *This design description should be in **sufficient detail** that another team of engineers can look through it and implement it.*

A past group has implemented the project with a neural network. We will implement it with a graph convolutional neural network. This will satisfy the functional requirement of using a graph neural network. The previous group encountered issues with accuracy in their translations, and the implementation of a GCN will also help in meeting that requirement. Based on our research, the best way for us to meet the accuracy requirement is to run the natural language through 2 different encoding methods. First, it will be put through a bidirectional LSTM layer so that each word will incorporate information about words that are close to it in the sentence. Second, it will be put through 2 layers of GCNs which will incorporate details about each word's syntactic and/or semantic dependencies within the sentence.

Because the client requested that we use Spektral to develop the neural network, we will have a server running on a remotely hosted linux VM that manages all interactions with the neural network.

None of the standards are directly related to our project at the current moment but as the design and implementation phase continue, standards could possibly come into play.

We plan to follow industry standards by documenting our work thoroughly, using a modular design, and making a simple UI.

3.4 Technology Considerations

Strengths

For the user interface we are using IntelliJ. The strengths of this is that intelliJ is has an abundance of documentation and a large community that uses and helps other users. Creating, using, and installing IntelliJ plugins is really simple for both developers and users. We also looked into using eclipse but after much discussion with the client and advisor, it was highly suggested that IntelliJ would be the preferred IDE to work in.

For our neural network, we are using Spektral. Spektral is nice because it basically lays down the groundwork for what we need to do with neural networks. It is also one of the limited libraries that we are able to use when working with neural networks.

Weaknesses

As far as IntelliJ goes, one of the weaknesses is with the plugin. The plugin does not allow the user to modify everything with the code editor. This problem was solved by the previous group by modifying parts of the features and determining other methods of adding the features outside of the IntelliJ plugin.

One of the weaknesses of using Spektral is that it does not run on Windows. Spektral only runs on linux and MacOS and to get around this we are using linux virtual machines to be able to run Spektral on our devices. Another weakness is that Spektral is a pretty niche program. While useful to our project, there are not many other projects that use Spektral open source. This means that there are not many examples of how to use Spektral and everything we develop is basically from our own design using experiments of our own.

3.5 Design Analysis

As of now, we anticipate that the proposed design from section 3.3 will work with hopefully minimal setbacks. We will have to modify all current code directly involved with translation from natural language to working code as well as create a working graph neural network with minimal external tools, but our team is learning more about graph neural networks each day and are becoming more confident that this solution will be feasible within the school year.

If modifications to the design are required, it will likely be with our integration of the LSTM layer with the graph neural network. We are currently in the process of researching how to get the two layers to communicate with each other, and the best implementation of this communication is subject to change as we learn more about each respective subject.

Otherwise as new, unforeseen challenges arise, obviously we will modify/iterate the design in any way we feel works best for the successful development of the project. All necessary changes will be approved by the client.

3.6 Development Process

Our team will follow an agile development process with two week sprints and weekly client meetings to discuss any status updates and necessary changes. We chose agile because of its iterative nature and ease of adaptability since we do anticipate unforeseen challenges with the project in the future. We will likely also break out into two teams for development and integration to make better use of our time.

Milestones have also been created to better keep track of where the team is in its development schedule. The major milestones include full completion of the graph neural network on the server, its integration with the version one plugin, and sufficient unit testing with various input strings. As errors arise in testing, the agile development process should allow simple modifications to the project code whenever necessary.

3.7 Design Plan

Describe a design plan with respect to use-cases within the context of requirements, modules in your design (dependency/concurrency of modules through a module diagram, interfaces, architectural overview), module constraints tied to requirements.

Firstly, the previous group struggled with accuracy in their natural language translations. Ideally, with our creation of the functional graph neural network layers, this insufficiency in accuracy will be remedied, so that is our first big design milestone. To accomplish this, we chose to use a library

with helpful built in functionality called Spektral, which we need to run on a linux virtual machine since Spektral does not support Windows at this time.

Once we have created the graph neural network layers using Spektral, we can then move on to creation of the LSTM layer. This layer will likely be easier to implement than the others, but getting the LSTM layer to communicate with the graph neural network layers will be the tricky part, and is subject to change.

After communication is established between the LSTM layer and the graph neural network layers, we can begin integrating these translation layers into the version one code from last year's team to time effectively make our server usable via an Intellij plugin. More research into Intellij plugin functionality may be necessary at this point.

If we have extra time before testing once integration is complete, our team may look into creating a new LSTM/graph neural network model to search for common programming errors. This will likely take a large amount of time and will extend the testing process, but it will also make our plugin much more marketable to consumers if feasible with our time constraints.

Finally, after all prior steps have been completed, testing can begin. Most testing will involve examining the input and output of the graph neural network and making small modifications to the code based on those results. More details on the testing phase can be found below.

4 Testing

Most of our testing will revolve around testing the graph convolutional network(GCN) for good output in the beginning of our project. This will be dependent on how much we can train the GCN and how well it is created. We will also have to test the implementation of Keras, an API that we plan on using to convert our natural language text into numerical values that the GCN can work with.

For testing the natural language translation we can test using known input and relating it to a desired output and the same with Keras for pre-processing. We will also have to test some post-processing functionality to clean up the GCN's output into usable code.

Beyond the language translation we will have to test our pugin's ability to recognize and suggest corrections to common code errors.

4.1 Unit Testing

- Neural Network Model

After we develop and train our neural network, it will produce a model that we will use for the actual translation process. We will then run accuracy tests on the model to determine the accuracy of its predictions, and update the model if necessary to increase accuracy.

4.2 Interface Testing

- Server-Model Interface

The only function of the server is to pass text into the model and retrieve the translated text, so the connection is important to test. To test this interface, we will make a test model that does nothing to the text (the text that goes in is the same as the text that goes out), and compare the end text to the start text to ensure that things are being passed properly.

- Plugin-Server Interface

The plugin will need to send text to the server for translation, and it will need to receive the translated text back to use later. To test the plugin side of this connection will be tested by having a locally-hosted server to receive text to make sure it's being sent properly, and send the text back to make sure the plugin can receive a response properly. The server will be tested in a similar way by having a program on the server VM send messages to the server to make sure it can receive text properly, and then receive the translated text back to make sure text is being sent properly.

4.3 Acceptance Testing

For our project's acceptance testing we will be showing off the plugin's ability to translate natural english into functional java code to our client. From there if it is not accurate enough for the client we could train the model more to increase its accuracy.

For the code correction suggestions our project will have to routinely and correctly identify and correct common code errors. To involve our client in this step we would show off the plugin's ability to help fix these common errors.

4.4 Results

– List and explain any and all results obtained so far during the testing phase

- Include failures and successes

- Explain what you learned and how you are planning to change the design iteratively as you progress with your project
- If you are including figures, please include captions and cite it in the text

We have not started testing yet. Testing will begin within the next 2 weeks.

5 Implementation

Describe any (preliminary) implementation plan for the next semester for your proposed design in 3.3.

6 Closing Material

6.1 Conclusion

Summarize the work you have done so far. Briefly re-iterate your goals. Then, re-iterate the best plan of action (or solution) to achieving your goals and indicate why this surpasses all other possible solutions tested.

6.2 References

List technical references and related work / market survey references. Do professional citation style (ex. IEEE).

6.3 Appendices

Any additional information that would be helpful to the evaluation of your design document.

If you have any large graphs, tables, or similar data that does not directly pertain to the problem but helps support it, include it here. This would also be a good area to include hardware/software manuals used. May include CAD files, circuit schematics, layout etc,. PCB testing issues etc., Software bugs etc.