Intelligent Code Editor with Code Suggestion and Code Correction

FINAL REPORT

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: 04/25/21

Table of Contents

1. Evolution of Design

1.1 Plugin Updates

The plugin originally was intended to be made as a plugin for Intellij. After discussing with our client and updating our needs, we found it best to move towards a Visual Studio Extension. This allowed us to work without the need of a server and a more friendly environment with python. Visual Studio also supports C++, matching the data set that we found to train our model. Aside from a new editor, the rest of the plugin design remained the same.

1.2 Python Updates

We had originally planned to develop a completely custom NMT system using the Spektral library, but found that it was not flexible enough to suit our needs. From there, we discussed options with our client and came to the conclusion that using an already existing neural network would give us the best possible results within the timeframe of our project. It was decided that we would instead use a classification neural network because it would give us higher accuracy with basic code syntax, especially since there are a fairly limited number of types of lines of code.

1.3 Server Updates

After discussing with our client we decided to find other ways to implement our python. The original plan was to go with a server and then require the user to constantly have access to the internet. After some research we found that we could accomplish implementing python without the use of a server through using IronPython. IronPython allows us to run python scripts through our code without the use of a server. After some experimentation with IronPython, we found that it was out of date, and incapable of running the python that we needed. This led us to create a new locally hosted server, on which we run the classification model. Since the server is locally hosted and the model is trained, this allows the classification to run without internet access.

1.4 Data Updates

The data we use for training our graph neural network has changed significantly so we can get a higher accuracy out of the graph neural network. This change came in the form of sourcing a much larger dataset. However, with this larger dataset we needed to move from a java code center to C++ as it was the language the data set had. The new data also had to be sorted into classification fields so the model can learn properly.

2. Requirements

2.1 Functional Requirements

- Ability for user to input code and natural language into an editor
- Translation should be fast as to not annoy the user
- Ability for plugin to classify and translate natural language to code
- Translation should only be initiated by the user
- Ability for code to be executed once translated
- Capability for multiple natural language statements to be translated to code at once

2.2 Non-Functional Requirements

- Must have translational accuracy greater than 50%
- Can give 1 - 3 different possible translations
- The translation process should take less than 5 seconds
- Should be usable from anywhere - i.e. without internet access
- The plugin should be easy to use once added to the IDE

2.3 Technical Requirements

- Must use Visual Studio Extension development
- Spektral will be used for it's machine learning libraries
- Graph Convolutional Networks will be used for natural language to code translation

3. Industry Standards

3.1 IEEE Standards

- IEEE P14764
    - This standard is about software maintenance. It uses a process model to discuss and depict aspects of software maintenance using criteria established to apply during the development of the software and during the execution of the software. The purpose of the standard is to consistently plan for maintenance throughout the lifetime of a project from development to execution.
- IEEE 29119-2-2013
    - This standard is about software testing. It is a universally agreed upon standard to be used by anyone when testing software. The standard supports all kinds of testing and is used in the development life cycle. The purpose of the standard is to have a set of standards for testing since testing is key to risk-mitigation in software development.
- IEEE P15026-2

- This standard is about assurance. It does not set a standard on the quality of the contents but rather on the structure and its assurance cases. This allows for precise use of terms and limits inconsistencies and subject use of certain terms. The purpose of the standard is to have consistencies and precise language in assurance cases.

3.2 Other Standards

- Agile Workflow
- Test Driven Development
    - Both of these standards are common in regards to this type of project and we followed them to the best of our ability while working.

4. Engineering Constraints

4.1 Constraints

- Had to be completed in 2 semesters
- Had to be completed with no budget

5. Cyber Security

5.1 Security Concerns

While we initially were going to use a remote server which could have caused security issues, we ended up locally hosting our server which means that the plugin is as secure as the network the computer is on. We are not taking any data from the user and are only using the text passed in by the user.

6. Implementation

6.1 Visual Studio Extension

Visual Studio Extensions are implemented through C# and .NET. Having never used C# before and having not created an extension, this was new to us. However, C# is very friendly and has many tutorials regarding its implementation and uses with other languages. Visual Studio has great template projects for creating an extension. The plugin is easily accessible from the context menu in the text editor. When selected, the plugin passes the highlighted text to our translation model one line at a time, and replaces each line in the text editor with the resulting translated code.

6.2 Python Implementation

The Python implementation of this project can be broadly divided into 2 parts: the neural network, and the training data preprocessor. The neural network that the project is based around was originally developed for the paper "Graph Convolutional Networks for Text Classification" (see related literature), and was adapted for use with our project. It is a classification GNN based off of the TensorFlow library.

The training data preprocessor was developed to be used with the SPoC dataset originally developed for the paper "SPoC: Search-based Pseudocode to Code" (see related literature). The dataset contains roughly 300,000 lines of pseudocode and its equivalent code. The preprocessing system uses regex to parse the code and determine what classification each line of equivalent pseudocode should fall in. It then separates the data based on the classification and places them into files to be used for training by the neural network.

7. Testing

7.1 Data Parsing

As training our graph neural network came closer, it became apparent that we needed our training data to be the correct form for training. This required that we are able to classify the data used for training and sort it into appropriate categories. For sorting, we leveraged a python library called Regex. Testing the sorting of the filters for Regex was done by https://regex101.com/. This site allowed us to quickly test our filters on strings. Finding missing filters was done by adding a catch for anything that is not sorted. This allowed us to add the sorting for lines that do not filter properly.

7.2 Classification Testing

Testing for the neural network itself was done using some built-in systems in TensorFlow. Each stage of the training process produces an output stating the current loss and training/validation accuracy after it completes, and a final statistical block is produced by the model after the training is completely finished. The final accuracy of the network was calculated to be roughly 97%, although some additional inaccuracy is introduced in the process of converting those classifications to actual code.

	precision	recall	f1-score	support
0	0.0000	0.0000	0.0000	23
1	0.9400	0.9732	0.9563	2577
2	0.0000	0.0000	0.0000	3
3	0.9906	0.9832	0.9869	11530
4	0.0000	0.0000	0.0000	5
5	0.8805	0.9352	0.9070	386
accuracy			0.9780	14524
macro avg	0.4685	0.4819	0.4750	14524
weighted avg	0.9766	0.9780	0.9772	14524

8. Related Products and Literature

8.1 Related Literature

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, & Maosong Sun. (2019). Graph Neural Networks: A Review of Methods and Applications.

Yong Cheng, Wei Xu, Zhongjun He, Wei He, Hua Wu, Maosong Sun, & Yang Liu. (2016). Semi-Supervised Learning for Neural Machine Translation.

Guillaume Lample, Alexis Conneau, Ludovic Denoyer, & Marc'Aurelio Ranzato. (2018). Unsupervised Machine Translation Using Monolingual Corpora Only.

Miltiadis Allamanis, Marc Brockschmidt, & Mahmoud Khademi. (2018). Learning to Represent Programs with Graphs.

Liang Yao, Chengsheng Mao, Yuan Luo. "Graph Convolutional Networks for Text Classification." In 33rd AAAI Conference on Artificial Intelligence (AAAI-19), 7370-7377

SPoC: Search-based Pseudocode to Code from Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, Percy Liang

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.

8.2 Related 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.

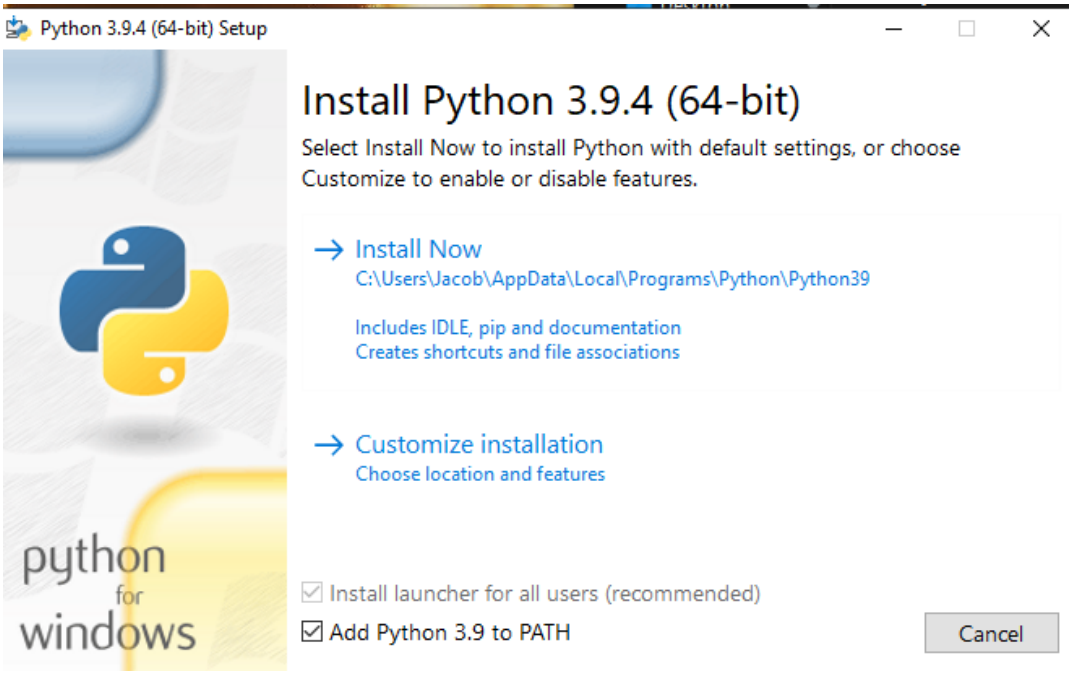
9. Appendices

9.1 Operation Manual

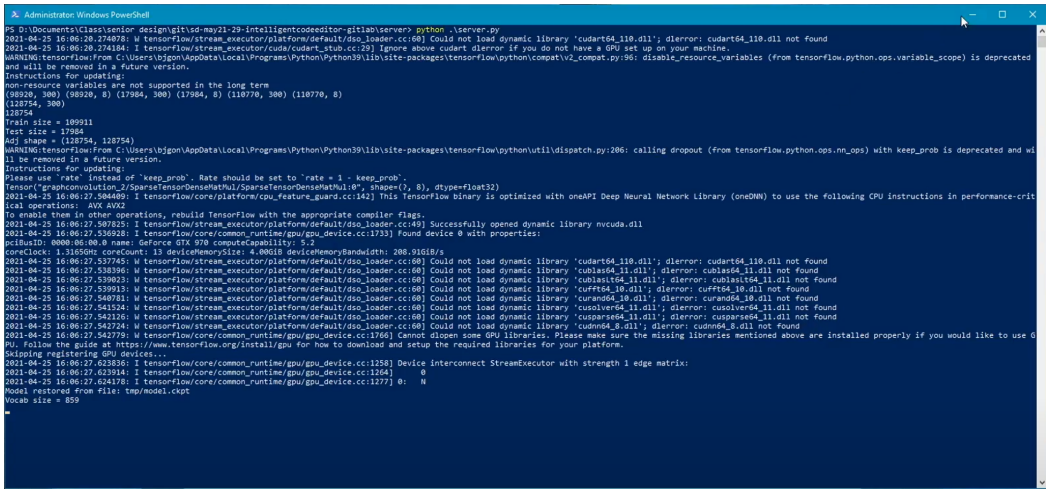
Steps for setup:

1. git pull (master branch)
    a. Close git & disconnect from VPN

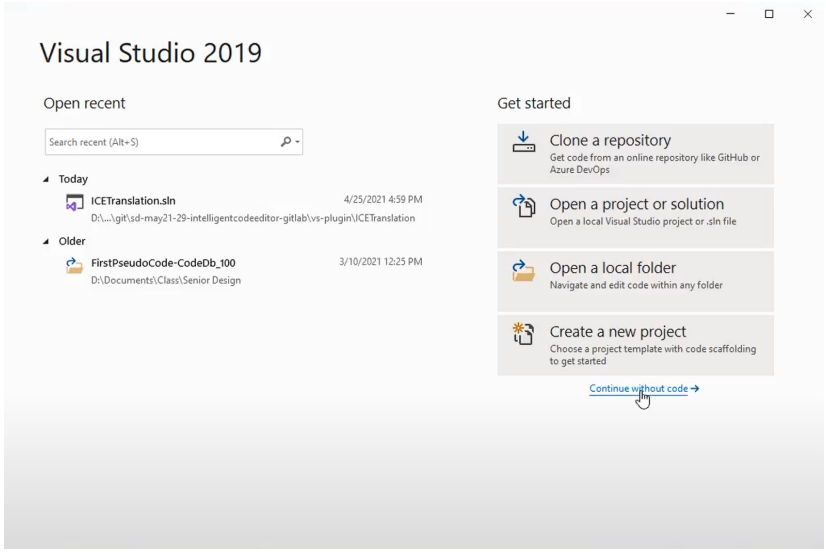
2. install 64-bit python
    a. uninstall 32-bit python (Optional/Troubleshoot)
3. Add python to path



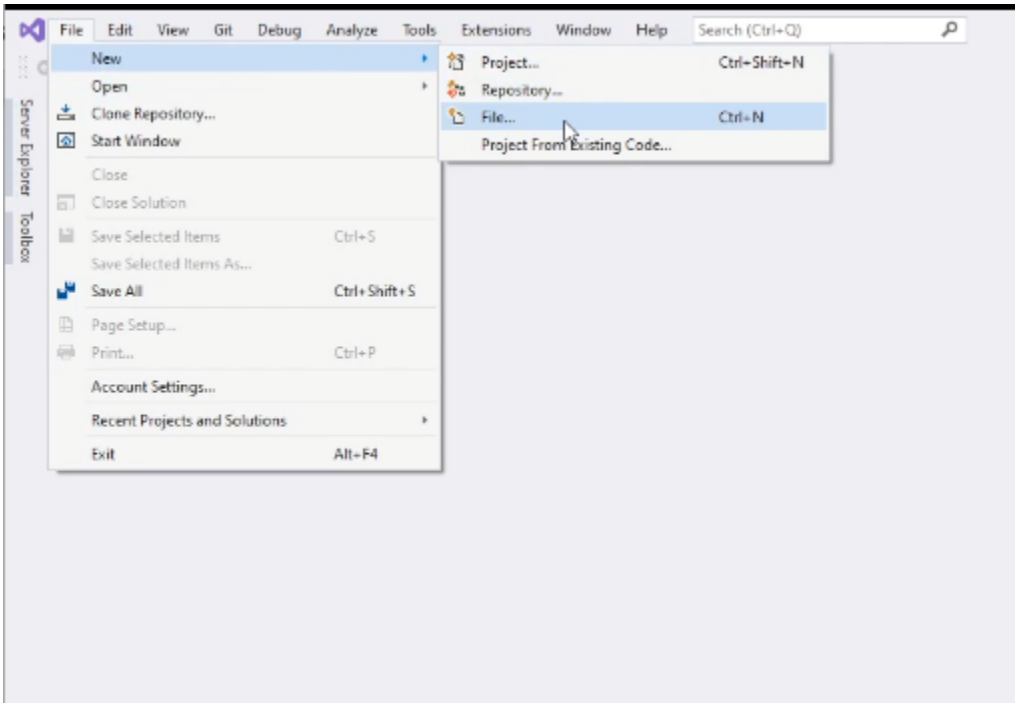
4. run powershell as administrator
    a. pip install --upgrade pip
        i. ignore the red text
    b. pip install tensorflow
    c. pip install sklearn
    d. pip install networkx
    e. pip install pathos
5. Open File Explorer
    a. Navigate to the VS_plugin/ICE_extension/ICE_extension/bin/release folder in the git
    b. run ICETranslation.vsix (This downloads the plugin to your Visual Studio
    c. Navigate to sdmay_...gitlab/server
    d. run server.py



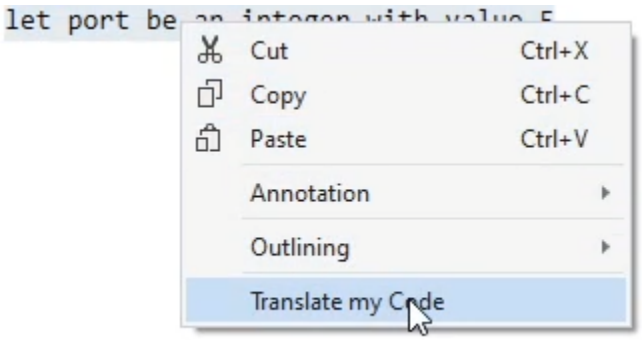
6. Open Visual Studio
    a. select “Continue without code”



    b. file -> new -> file -> text file



c. type “let port be an integer with value 5”
d. highlight the text and right click on it
e. select “Translate my Code”



7. (wait 1 minute) *Hurray!*

9.2 Alternative/Initial Versions of Design

The initial version of the design involved both an internet server and Intelli-j as the desired plugin. After a few meetings with our client and advisors we had decided to switch over to Visual Studio. The main reasons behind this was we believed that if we needed to have a server it would be much easier to implement in Visual Studio as its extension program was much friendlier than that of Intelli-j's.

We also found that Visual Studio is much friendlier working with other code so if we found it possible to add the python code directly to extension we would look towards that path. After some extensive research, we found that it may be possible to add in python code with the use of IronPython in the extension. We managed to get some low level scripts to work in the extension but found there to be some glitches when working with our complicated models. After discovering these glitches, we decided to go to (our final design) which is a locally hosted server.

9.3 Other Considerations

Over 200 hours were spent this semester on the implementation and update of the project. This does not include additional research on topics regarding the project along with small and minor updates throughout the project lifetime. This is specifically for this semester of the project and does not include the initial semester of working on the project along with the winter break where progress was also made. The time also does not include meeting times with team members and the client and advisor. The purpose of this section is to show future members of this project the minimum amount of time we had dedicated to this.

Refer to the design document for additional information regarding specifics on the project throughout its lifetime.