Improvement of Source Code Conversion for Code Completion

Mika Turk
m.j.turk@student.tudelft.nl

Supervisor Maliheh Izadi
Professor Arie van Deursen
Advisor Georgios Gousios

1 Introduction

Code Completion is the task of using the source code a developer has already written and predicting what they will write next. Having a computer predict what a developer will write next instead of them having to type it themselves saves them time if the suggestions are good and appear quickly. CodeFill has introduced a conversion from python source code to token sequences, this makes it possible for CodeFill to predict types [1]. Converting the content of the file the developer is working on is crucial as this influences the total latency from the developer typing something to the suggestion appearing on their screen. Having as little errors as possible is also important as an error in conversion results in no suggestions at all.

2 Approach

To uncover slow parts of the conversion, we decided to use a program that outlines on which lines the most time is being spent, also known as a line profiler. To optimize the performance, we will look at when to use the python library "pandas" and when to use python's built-in data structures like list and dictionary for storing and processing data.

To evaluate our improvements, we will be using the CodeFill dataset. The part of the dataset we will be using consists of four separate datasets, in 4 different orders of magnitude: 400-600, 4K-6K, 40K-60K, and 400K-600K bytes.

To convert JavaScript source files into token sequences, we used the js-tokens package to tokenize the source files into lists of tokens, process this tokenized form, and produce a new token sequence file.

Figure I

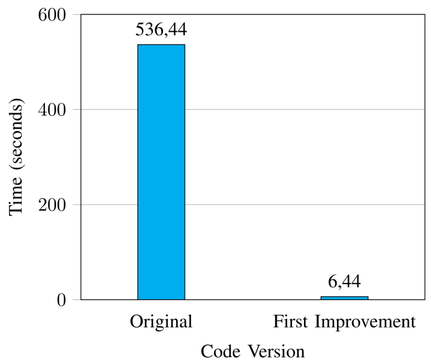

Figure II

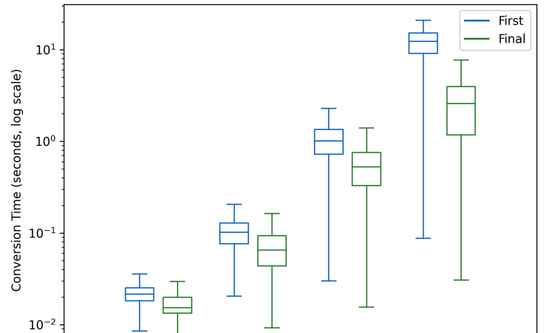

Figure III

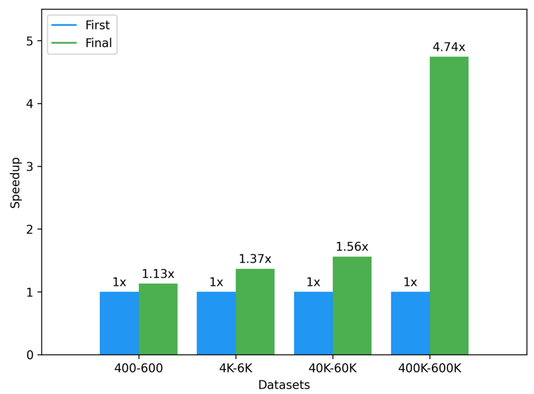

Figure IV

```
import { join } from 'node:path'

/**
 * Says hello
 */
function sayHello() {
  console.log(`Hello world! ${1+2}`);
  // Uses built-in path.join
  console.log(join('./hello', 'world'))
}

sayHello();
```

```
IdentifierName Punctuator IdentifierName
Punctuator IdentifierName StringLiteral
IdentifierName IdentifierName Punctuator
Punctuator Punctuator
IdentifierName Punctuator IdentifierName
Punctuator TemplateHead NumericLiteral Punctuator
NumericLiteral TemplateTail Punctuator Punctuator
IdentifierName Punctuator IdentifierName
Punctuator IdentifierName Punctuator
StringLiteral Punctuator StringLiteral Punctuator
Punctuator
Punctuator
IdentifierName Punctuator Punctuator Punctuator
```

3 Results

After improving the python conversion by replacing the original DataFrame creation with a much more efficient one, the conversion time for a 77 Kilobyte file went from 536,44 seconds to 6,44 seconds, a speedup of 83x which can be seen in Figure I.

With more refinements, we eventually got the conversion function to be even faster than with just this first improvement.

After running the 4 datasets through the function after the first improvement and the final conversion function, we obtained the time it took each function to process each dataset, they can be seen in Figure II. We also computed the average speedup for each dataset which can be seen in Figure II.

We created a simple JavaScript conversion that converts JavaScript into token sequences, example input and output can be seen in Figure IV.

4 Conclusion

The new conversion functions speed up the process significantly, this allows the use of even larger datasets which can be used to achieve even greater accuracy. The changes in the conversion also resulted in a higher proportion of files being available to process, the 400K-600K dataset went from 307 to 446 out of 500, a 45% increase.

[1] M. Izadi, R. Gismondi, and G. Gousios, "Codefill: Multi-token code completion by jointly learning from structure and naming sequences," 2022.