Towards Intelligent Error Prediction and Rectification in IDEs

Pritish Kishore Kumar

1 Introduction

Code prediction is a technology that is primarily used in autocomplete techniques in intelligent Integrated Development Environments (IDEs). Popular IDEs are language-specific offerings from JetBrains such as IntelliJ (Java intensive), PyCharm (Python intensive) etc. or Eclipse. Autocomplete is very useful to developers as it serves to ease their effort in writing good code, one important use case being the prediction of the next code token that the developer could potentially use. Another use case is that it does away with the requirements for developers to remember API calls and function names.[2]
However, while autocompletion serves to ease a developer's task of writing good code, this unfortunately only applies to typing syntactically or contextually correct code. If the code being typed is contextually or syntactically incorrect, then the autocomplete feature obviously doesn't offer up any suggestions. Some IDEs however offer some intuitive indicators towards errors such as:

- Underlining or demarcating such errors in red.
- Offering tooltips which pop-up upon hovering over such demarcated text/errors.

The utility of these indications are however quite limited: they merely inform the user that an error exists or that a particular symbol or token can't be resolved. These are not as helpful or intuitive as they were perhaps envisioned.

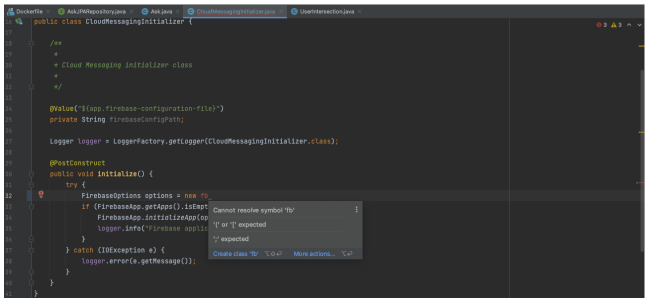
2 Proposal

As mentioned in the previous section, intelligent error correction suggestions and/or mechanisms currently offer very limited and counter-intuitive options in IDEs. With the vision in mind to make these mechanism more usable and intuitive, we propose to enhance these error rectification mechanisms with AI enabled suggestions. The two use-cases listed below should serve to illustrate our proposal:

- **Context based suggestions and Transformer based result ranking**: Certain IDEs offer tooltips which pop-up upon hovering over such demarcated text/errors. The current error tooltip (for example in Intellij) is rudimentary at best, providing the user only an indication that an error exists, but very little information about the actual cause the error. This is illustrated by **Fig 1**. In the figure, the error occurs when the right side of the assignment statement is syntactically incorrect. However, the tooltip only indicates that erroneous token can't be resolved.
  This tooltip could possibly be enhanced with context based suggestions to help resolve the unresolved token *'fb'*. In this case, since the resolution error occurs after the token *'new'*, from context it is clear that the left side of the statement expects a new instance of the class *'FirebaseOptions'*. The tooltip could therefore suggest either a constructor OR a builder instance, whichever is available for the current context; ie; *'FirebaseOptions'*.

SE4AI WiSe'21, Winter Semester 2021, Darmstadt, Germany
2021.

- **Context based StackOverflow solution suggestions for runtime errors**: When a syntactically correct program (which also compiles correctly) produces a runtime error on execution, developers instinctively seek out the stacktrace of the program run, copy a part of it and search for the error on Google. This typically results in list of StackOverflow links or blogs which a developer would then start perusing in an attempt to resolve the error.
  We propose to integrate this functionality into an IDE in an attempt to minimize the effort on the developer's part. For the sake of visualization, a tooltip/dialog box which could appear as a context-menu of sorts on selecting a portion of the stacktrace and right-clicking on it. This would the contain StackOverflow link suggestions which relate to current context or project within which the developer is working.
  We postulate that this would not only minimize the effort of the developer in searching for the appropriate links, but the intelligent introduction of context into the StackOverflow search process would further help refine appropriate results.



```
public class CloudMessagingInitializer {

    /**
     *
     * Cloud Messaging initializer class
     *
     */

    @Value("${app.firebase-configuration-file}")
    private String firebaseConfigPath;

    Logger logger = LoggerFactory.getLogger(CloudMessagingInitializer.class);

    @PostConstruct
    public void initialize() {
        try {
            FirebaseOptions options = new fb
            if (FirebaseApp.getApps().isEmpt
                FirebaseApp.initializeApp(op
                logger.info("Firebase applic
            }
        } catch (IOException e) {
            logger.error(e.getMessage());
        }
    }
}
```

```
Cannot resolve symbol 'fb'
'(' or '[' expected
';' expected
```

Figure 1. Unhelpful tooltip for syntax errors while typing in IntelliJ

The following sections are dedicated to exploring each of these ideas in sufficient detail.

2.1 Context based suggestions and Transformer based result ranking

This proposal envisions enhancing with context based suggestions the tooltips currently displayed by IDEs on erroneous tokens, to aid the developer with faster error resolution. As illustrated in **Fig 1**, the tooltip only indicates that erroneous token can't be resolved, and which could possibly be enhanced with context based suggestions, such as either a constructor OR a builder instance, whichever is available for the current context; ie; *'FirebaseOptions'* to help resolve the unresolved token *'fb'*.
In order to do this, we would require a dependable token recommendation system which takes the current context into account. The most popular next token recommendation techniques utilize popular machine learning models such as Type-based inference or sequential Recurring Neural Network (SeqRNN) ones. However,

as summarized comprehensively in [2], there exist major limitations in those two: Type-based models such as Jedi[1] or Eclipse[2] recommended and rank the next token in a very naive way (usually alphabetically), while their research has showed very clearly that popular machine learning based code prediction models such as SeqRNN recommends the next token in such a way that the correct result is present in approximately the top 2.7 results, leaving a definite margin for improvement.[2]

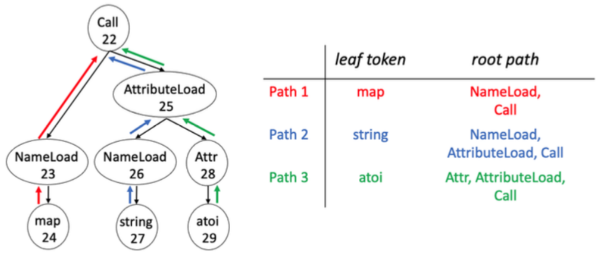
2.1.1 Transformer based intelligent next-token recommendation. While the SeqRNN ML model yields impressive results in terms of next token recommendation, using Transformer-based models for next token prediction (instead of RNN) improved the ranking statistics: Correct results were displayed to developers within the first 1.5 - 2 results, an improvement of about 0.5 to 1 ranks higher.[2]
A Transformer is a machine learning model first introduced by **GoogleBrain**.[3] It is a type of model which processes input as a whole (for instance an entire sentence) and does not depend on past hidden states to retain past information (like Recurrent Neural Networks (RNN) do).[4] This has the advantage that Transformers don't risk losing past information.
An abstract syntax tree (AST) is generally defined as the abstract syntactic representation of source code.[1] Within this representation, each node of the AST represents a construct/token of the source code. The research in this paper postulates the above mentioned improvement in accuracy by feeding ASTs into Transformers. However, since Transformers are sequential models, only partial structures of the AST can be fed in at one point in time. Their solution to this was to explore three different methods to this end:[2]

- Decompose source code into partial programs and consequently each partial program into a sequential stream of source code tokens (SeqTrans)
- Decompose the AST into multiple paths and subsequently feed each path into the Transformer (PathTrans)
- Decompose the AST into its various traversal orders and subsequently feed each traversal into the Transformer (TravTrans)

2.1.2 SeqTrans. Since a Transformer is designed to take only sequential input, the input for this particular method works out of the box with the Transformer. A partial program (which is effectively a sequence of source code tokens) is fed as input to the Transformer, which in turn is expected to yield a single source code token as output.This method was already proven to be better than SeqRNN, as well as provided a platform for the following two AST based methods.

2.1.3 PathTrans. This method works on the concept of root paths: A root path is the path from a leaf node of an AST to to its root. Such an example is depicted in Fig 2 If an AST has multiple leaf nodes, there will be multiple root paths. Since each root path forms a sequence of internal nodes of the AST to each leaf token, it would ideally capture all the syntactical information along this path. Consequently, the root paths help in leaf token prediction when they are fed into the Transformer.

[1]https://github.com/davidhalter/jedi
[2]https://www.eclipse.org/pdt/help/html/working with code assist.htm
[3]https://research.google/teams/brain/



	leaf token	*root path*
Path 1	map	NameLoad, Call
Path 2	string	NameLoad, AttributeLoad, Call
Path 3	atoi	Attr, AttributeLoad, Call

Figure 2. Rootpath concept

2.1.4 TravTrans. This method works on the concept of tree traversals;ie;feeding a sequence of nodes of the AST in a certain tree traversal order. The popular tree traversal methods include pre-order, post-order and in-order and can be applied to tree structures such as AST. The method involves feeding code token sequences which are obtained from pre-order traversals (in Depth First Search order since pre-order traversal is a kind of DFS) of the AST to optimize leaf token prediction. Fig 3 represents an AST. If we consider

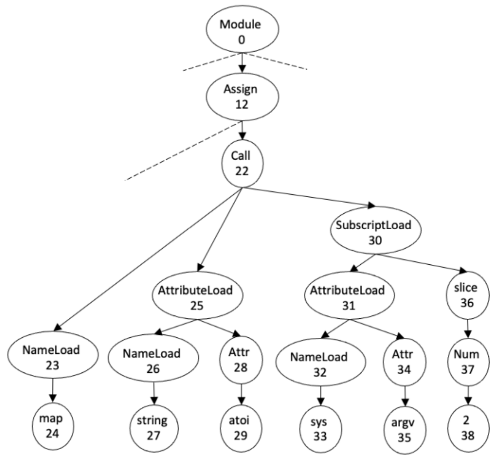


Figure 3. AST as input to TravTrans

node 29 for example (atoi), the the node chain in DFS order preceding it would be:
Call -> NameLoad -> Map -> AttributeLoad -> NameLoad -> string -> Attr

3 Envisioned implementation

The token prediction results are generally expressed in terms of **mean reciprocal rank** percentages. As mentioned earlier, for typical RNN based methods, this percentage is about **37**, while the most efficient method, **TravTrans**, which functions on feeding AST tree traversal sequences to Transformers, was found to increase this percentage by about **18**. Since 37 roughly equates to the correct token being present in the first 2.7 results, this 18 percent increase meant that now the correct token could be present anywhere between the first 1.5 and 2 results. As a consequence, we propose to use TravTrans as the method of choice for input transformation. Meanwhile, similar to the implementation [2], this implementation would also require a large dataset. The dataset would comprise of source code files from any popular language (Java, Python etc.) decomposed into ASTs (to later be fed into Transformers). It follows logically that an increase in the size and variety of the dataset would serve to improve the accuracy of the recommendation. Since it could potentially be tested internal to a team or organization, the dataset could largely comprise of AST decomposed from internal source code repositories. Since it is generally inadvisable to reinvent the wheel and because it produce a proven increase in MRR (Mean Reciprocal Rank), we propose to opt for for the GPT-2 small implementation[3] based on the PyTorch library[4], similar to what was utilized in the research.[2]. The schematic illustration of such a GP2 Transformer is depicted in Fig 4

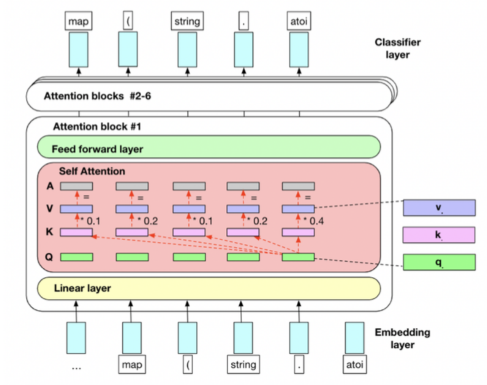


Figure 4. GP2 Transformer

References

[1] Baojiang Cui, Jiansong Li, Tao Guo, Jianxin Wang, and Ding Ma. 2010. Code Comparison System based on Abstract Syntax Tree. In *2010 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT)*. 668–673. https://doi.org/10.1109/ICBNMT.2010.5705174

[2] et al. Kim, Seohyun. 2021. Code prediction by feeding trees to transformers. *IEEE/ACM 43rd International Conference on Software Engineering (ICSE)* (2021).

[3] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[4]https://github.com/graykode/gpt-2-Pytorch