Divinator: A Visual Studio Code Extension to Source Code Summarization

Rafael S. Durelli  
Federal University of Lavras  
Lavras-MG, Brazil  
rafael.durelli@ufla.br

Vinicius H. S. Durelli  
Federal University of São João del Rei  
São João del Rei-MG, Brazil  
durelli@ufsj.edu.br

Raphael W. Bettio  
Federal University of Lavras  
Lavras-MG, Brazil  
raphaelwb@ufla.br

Diego R. C. Dias  
Federal University of São João del Rei  
São João del Rei-MG, Brazil  
diegodias@ufsj.edu.br

Alfredo Goldman  
University of São Paulo  
São Paulo-SP, Brazil  
gold@ime.usp.br

Abstract
Software developers spend a substantial amount of time reading and understanding code. Research has shown that code comprehension tasks can be expedited by reading the available documentation. However, documentation is expensive to generate and maintain, so the available documentation is often missing or outdated. Thus, automated generation of brief natural language descriptions for source code is desirable and has the potential to play a key role in source code comprehension and development. In particular, recent advances in deep learning have led to sophisticated summary generation techniques. Nevertheless, to the best of our knowledge, no study has fully integrated a state-of-the-art code summarization technique into an integrated development environment (IDE). In hopes of filling this gap, we developed a VS Code extension that allows developers to take advantage of state-of-the-art code summarization from within the IDE. This paper describes Divinator, our IDE-integrated tool for source code summarization.


Keywords: learning, source-code summarization, deep-learning

1 Introduction
Understanding source code is taxing, hence several studies have investigated approaches to aid program understanding. Although proponents of agile approaches maintain that code should be the main documentation, previous research [9] supports the notion that a natural language description of a given piece of code can make it easier to understand and make changes to that code. As pointed out by Sommerville [13], ideally, software should be an amalgamation of code and its associated documentation. However, despite the benefits ascribed to writing these natural language summaries (e.g., helping programmers to quickly gain a better understanding of the purpose of a given method or subroutine), programmers tend to put off writing source code summaries or avoid it altogether to save time and manual effort [12, 17]. Therefore, in practice, the only available documentation is the source code. To make matters worse, when available, documentation is usually inconsistent and out of date, so before relying on the available documentation, programmers have to make sure it is complete and up-to-date.

Researchers have tried to mitigate the aforementioned problems by automatically generating comments. Specifically, the task of trying to comprehend and generate summaries directly from source code has been aptly termed source code summarization [2]. Automatic source code summarization has been an active research area over the last decades [1, 6–8, 15, 16] given that by perusing succinct descriptions (i.e., summaries) of source code fragments developers can reduce the amount of code that needs to be read and understood. Essentially, code summarization is grounded in the idea that a brief natural language description of code (e.g., “Determines if an integer is even”) allows developers to understand the code’s purpose without having to read the code, saving them time from having to go over implementation details. Therefore, by relying on these code summaries, a developer can quickly filter out code that is pertinent to the task at hand.

In this paper, we present Divinator: a Visual Studio Code (VS Code) extension that allows developers to generate natural language descriptions from source code. More specifically, our extension utilizes CodeTrans [3], which is an encoder-decoder pre-trained model for Java, SQL, Python, and CSharp, to generate natural language summaries from chunks of code, methods, and classes. The remainder of the paper is organized as follows. Section 2 presents background information on source code summarization and outlines the main elements of Divinator. Section 3 provides a brief overview of the existing literature on code summarization. In Section 4 we further elaborate on the architecture of the proposed VS Code extension. Section 5 shows an example of how to use Divinator. Section 6 presents concluding remarks.
2 Background

As mentioned, during development and maintenance tasks, developers spend most of their time reading and understanding code. Being able to rely on a short description about the code’s purpose allows developers to save time and makes development and maintenance tasks less taxing on developers. However, research has shown that developers tend to avoid the manual effort of writing summaries themselves [12, 17], so automatic source code summarization has recently been gaining increased attention as a desirable alternative. Researchers have drawn from machine learning (ML) techniques and more recently deep learning (DL) to improve the quality of the generated summaries. In fact, advances in DL have recently come to dominate the state-of-the-art performance in source code summarization. For instance, BERT [7], XL-NET [16], ALBERT [6], RoBERTa [8], GPT-3 [1], and T5 [15] have been extensively used for code summarization.

To perform source code summarization our VS Code extension employs CodeTrans [3], which is based on the encoder-decoder transformer architecture. Specifically, Elnaggar et al. adapted the encoder-decoder model proposed by Vaswani et al. [14] and the T5 framework implemented by Raffel et al. [15]. The resulting source code summarization model is able to process Python, SQL, and CSharp code snippets. The dataset used to train the model was generated by Iyer et al. [5]. Such dataset was extracted from StackOverFlow\(^1\) and comprises code snippets from accepted answers with exactly one code snippet, and the target summarization is based on the title of the question.

3 Related Work

Software developers spend a great deal of time reading and understanding code that is poorly-documented, written by other developers, or developed using differing styles. During the past decade, researchers have investigated techniques for automatically documenting code to improve comprehensibility. Early work in the area of source code summarization used Natural Language Processing (NLP) techniques and heuristics, for example, exploring the frequency with which words were cited in code or the traditional Term Frequency Inverse Document Frequency method. Haiduc et al. [4] were the first researchers in the use of NLP techniques for source code summarization. Rodeghero et al. [11] evolved the work developed by Haiduc et al. [4], being incorporated into the eye movement analysis of programmers, where they first analyzed the signatures of the methods and then analyzed their implementation.

Moreno et al. [10] used stereotyping to assist in the technique of creating comments for class implementations in the Java programming language, where the type of class or the type of class methods, constructors, static methods were analyzed, among others, to aid in summarization.

Iyer et al. [5] pioneered the use of neural networks for source code summarization, in which a Recurrent Neural Network (RNN) and a Long short-term memory (LSTM) were used. Since then, most works for source code summarization have used different deep neural network architectures due to the state-of-the-art results obtained [18].

According to Zhu and Pan [18], the most applied neural network architectures in the source code summarization task are the RNNs, the Convolutional neural network (CNN) and the Graph Neural Network (GNN). The authors highlight the RNN architecture for the good performance obtained by the systems that adopt it. In addition to these architectures, another highlight is the use of the Sequence-to-sequence (Seq2seq) architecture, which is widely used in automatic text translation techniques and was soon adapted by several authors for the task of source code summarization.

4 Architecture

Many aspects of the IDE can be customized and improved through the Extension application programming interface (API). VSCode was built with extensibility in mind to the point that many core features are built as extensions using the Extension API. Divinator was built using the Extension

\(^1\)https://stackoverflow.com/
API, so it integrates seamlessly into VSCode and is streamlined to help developers get a state of the art source code summarization feature within the IDE.

Our extension is based on a microservices architecture: source code summarization runs as an isolated process implemented through a FastAPI\(^2\) endpoint, which is hosted on Amazon Web Services EC2\(^3\). An overview of the architecture of our extension is shown in Figure 1. As shown in Figure 1, source code summarization is prompted by the user upon selecting a snippet of code for summarization. The JavaScript part of our extension then sends a JSON file containing the selected snippet of code and related metadata. Given that our

\(^2\)https://fastapi.tiangolo.com/

\(^3\)https://aws.amazon.com/ec2/
extension is able to process and summarize Java, Python, and CSharp code, metadata includes information about filename extension. An example of JSON file generated by our extension is shown in Figure 1. Based on the filename extension a server-side FastAPI implementation selects the pre-trained model (i.e., CodeTrans) that should be run in order to summarize the chosen piece of code. Source code summaries are returned as JSON files (Figure 2).

Divinator also includes a local history option that allows developers to track all snippets of source code that have already been summarized as well as the corresponding summaries.

5 Divinator: Installation and Usage

To install our extension, users need to access the Extensions view from within VS Code by clicking on the Extensions icon in the Activity Bar on the side of VS Code. Upon typing “divinator” in the search box to filter the Marketplace offerings to extensions titled “divinator”, users should see our plugin in the list as shown in Figure 2. Clicking on the Install button will prompt VS Code to download and install the extension from the Marketplace.

There are two ways to use our extension: (i) by selecting the piece of source code to be summarized and right-clicking on the Divinator – Summarize Source-Code (Figure 3); alternatively, (ii) after selecting the chunk of code to be summarized, the user has to access the Command Palette (Cmd/Ctrl + Shift + P) and choose the Divinator – Summarize Source-Code option (Figure 4).

After selecting the Divinator - Summarize Source-Code option, our extension uses a POST request to send data to the server: a JSON file (as shown in Listing 1). Based on the information on the JSON file, the server-side code of our extension determines which pre-trained model should be used to summarize the given source code. As mentioned, this decision is mostly based on the filename extension sent to the server-side code. Upon processing the piece of code, the resulting summary (Listing 2) is sent back to the client VS Code as shown in Figure 5.
6 Concluding Remarks

Source code summarization has been studied for decades, however, only recently the area has started drawing great attention owing to how more recent techniques have improved the quality of the generated summaries by taking inspiration from methods in machine learning and recent advances in deep learning. Although many pre-trained models that have been fine-tuned for a myriad of software engineering-related tasks have become available, we believe that not much effort has been devoted to integrate these models into IDEs. As a first step towards bridging this gap, we designed and implemented Divinator. Through Divinator developers can take advantage of state of the art source code summarization feature from within the VSCode IDE, thus developers can easily leverage this feature during development and maintenance efforts. Our extension is able to generate natural language summaries for Java, Python, and CSharp. Divinator implements a microservice-based architecture that allows for scalability and mitigates the cost of model execution (on the server-side). As future work we intend to probe into how developers tend to use Divinator during development efforts, we believe such follow-up study can provide unique insights into how we can further improve Divinator.

References


