From Source Code to Natural Language: An Exploration of Code Summarization Techniques

Gelei Xu (gxu4@nd.edu), Ningzhi Tang (ntang@nd.edu)
University of Notre Dame, Computer Science and Engineering

Code, Data & Paper

Introduction

Summarizing code functionality in concise, natural language documentation can improve programmers' understanding the structure and logic of a code repository, thus enhancing software development efficiency.

However, programmers often neglect writing documentation due to time pressure and language barriers. Additionally, manually generated summaries are prone to being incomplete and outdated [1]. Thus, automatic code summarization has attracted increasing attention from the research community.

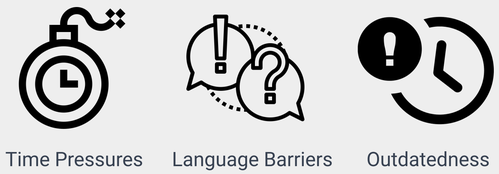


Figure 1. The challenges in writing documentation.

Traditional code summarization models include domain-specific methods and probabilistic grammars [2]. Current state-of-the-art models primarily use an encoder-decoder framework, encoding code tokens into vector representations and decoding them into natural language.

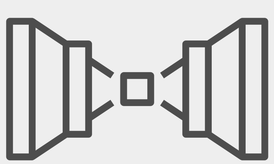


Figure 2. The encoder-decoder framework.

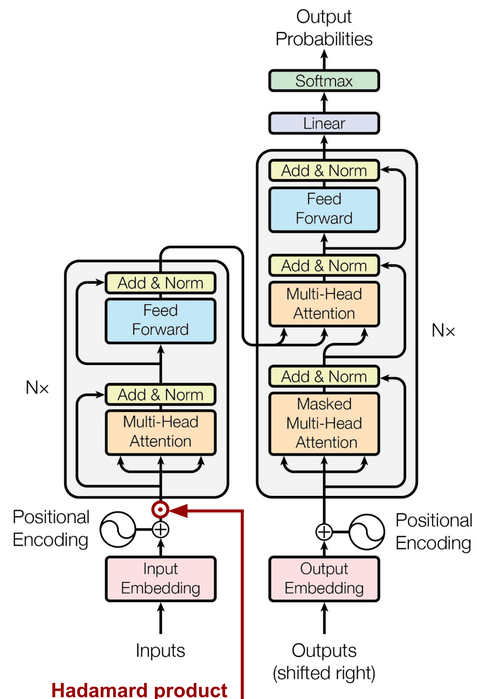
Source Code

```
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n - 1) + fibonacci(n - 2)
```

Natural Language

```
""" A function that returns the nth value in the Fibonacci
sequence using recursion . """
```

Listing 1. Example input and output of code summarization.



TF-IDF Encoding

Term Frequency - Inverse Document Frequency

$$w_{i,j} = tf_{i,j} \times \log(N/df_i)$$

$tf_{i,j} = \#$ occurrences of token i in document j

$df_i = \#$ documents containing token i

$N = \#$ documents

Figure 3. Our method's framework: vanilla Transformer [3] with TF-IDF encoding for input embeddings.

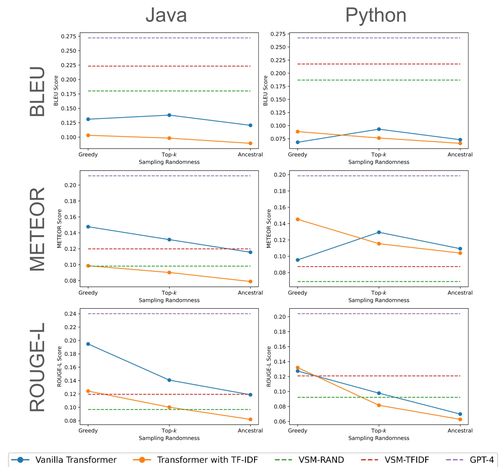


Figure 4. Experimental results on Java and Python datasets. VSM stands for Vector Space Model [4]. The *k* value in Top-*k* sampling is 100. GPT-4 uses five-shot learning from the training set as prompts, evaluating only 100 samples per dataset for convenience.

Target: `store a temporary file .`
VSM-RAND (Baseline): `[ path os config def`
VSM-TFIDF (Baseline): `1024 file target path tmp`
Vanilla Transformer (Top-*k*): `read the path of a file .`
Transformer + TF-IDF (Top-*k*): `return list of the file path .`
GPT-4: `stores a temporary file and returns its path .`

Listing 2. Comparison of our implemented methods' example summarizing outputs with the target.

Key Takeaways

KT1. Mismatch between word overlap metrics and human judgment.

KT2. High correlation between code and summary due to numerous identical tokens.

KT3. Integrating prior knowledge into model training is challenging.

KT4. GPT-4 remains state-of-the-art and appears to be a reliable standard.

Reference

[1] S. Haque et al. 2022. Semantic similarity metrics for evaluating source code summarization. In *ICPC '22*.
[2] T. H. Le et al. 2020. Deep learning for source code modeling and generation: Models, applications, and challenges. In *CSUR '20*.
[3] A. Vaswani et al. 2017. Attention is all you need. In *NeurIPS '17*.
[4] S. Haiduc et al. 2010. On the use of automated text summarization techniques for summarizing source code. In *WCRE '10*.

Acknowledgment: This project was completed for CSE 60657 Natural Language Processing, Fall 2023. Special thanks to instructor Prof. David Chiang and teaching assistant Aarohi Srivastava for their guidance. The opinions, findings, conclusions, and recommendations expressed are solely the authors' and do not necessarily represent the views of the Computer Science and Engineering Department at the University of Notre Dame.