Graph-Augmented Code Summarization in Computational Notebooks

April Wang[1] , **Dakuo Wang**[2*] , **Xuye Liu**[3] and **Lingfei Wu**[2]
[1]University of Michigan
[2]IBM Research
[3]Rensselaer Polytechnic Institute
aprilww@umich.edu, dakuo.wang@ibm.com, liux27@rpi.edu, wuli@us.ibm.com

Abstract

Computational notebooks allow data scientists to express their ideas through a combination of code and documentation. However, data scientists often pay attention only to the code, and neglect the creation of the documentation. In this work, we present a human-centered automation system Themisto that can support users to easily create documentation via three approaches: 1) We have developed and reported a GNN-augmented code documentation generation algorithm in a previous paper, which can generate documentation for a given source code; 2) Themisto implements a query-based approach to retrieve online API documentation as the summary for certain types of source code; 3) Themisto also enables a user prompt approach to motivate users to write documentation for some use cases that automation does not work well.

1 Introduction

Documenting the story behind code and results is critical for data scientists to work effectively with others, as well as their future selves [Zhang *et al.*, 2020; Kery and Myers, 2017]. The story, code, and computational results together construct a computational narrative. Data scientists find these computational notebooks particularly useful as they can support rapid *exploration* with code and *explanation* with natural language [Rule *et al.*, 2018].

Unfortunately, the ease of use of computational notebooks also comes with a cost. Data scientists often write fragmented and drafty code in computational notebooks during their quick experimentation of testing hypotheses or alternatives. It is a tedious process for data scientists to then manually document and refactor the raw notebook into a more readable computational narrative, thus many people neglect to do so [Wang *et al.*, 2021a].

Natural language processing (NLP) researchers have started exploring various ways to automatically generate code summarizations and documentation using source code as input [LeClair and McMillan, 2019; LeClair *et al.*, 2020]. But the documentation in data science notebooks are particularly challenging for two reasons: 1) documentations in a notebook may explain the rationale of a piece of code, or may interpret the outcomes of a table or a chart, which is difficult for existing automated code summarization techniques; 2) even for the documentation simply about what code snippet does, it is still challenging than other coding contexts, due to the unique nature of the notebook that one documentation markdown cell may cover multiple code cells.

In this paper, we propose Themisto, an end-to-end automated documentation generation system that can support users to create various documentations in a computational notebook. It has a frontend user interface that is integrated into the Jupyter Notebook environment (Fig. 1) thus users do not need to leave their familiar coding environment; its backend is built on top of a three-approach framework to tackle the two aforementioned challenges: 1) to generate the documentation that is not easy for an automation algorithm, we devise a query-based approach and a prompt-based approach so that the users can semi-automatically write documentations themselves; 2) to generate documentation for multiple code blocks, we consider each code block as a graph, and together as a hierarchical multi-graph structure and propose an attention-based hierarchical ConvGNN component to augment a seq2seq network to solve this problem. The HAConvGNN algorithm framework is reported in [Liu *et al.*, 2021].

In summary, we present a three-approach automated code documentation system to help Jupyter Notebook users to create documentation for their code and results. Builds on our previously reported HAConvGNN algorithm, the system has a human-in-the-loop design feature that can make the human users' work easier for the difficult-to-generate documentation.

2 Themisto

We design and implement Themisto as a Jupyter Notebook extension that supports data scientists to write better-documented computational narratives.

2.1 System Architecture

The Themisto system has two components: the client-side UI implemented as a Jupyter Notebook plugin using TypeScript code, and the server-side backend implemented as a server using Python and Flask.

*Contact Author

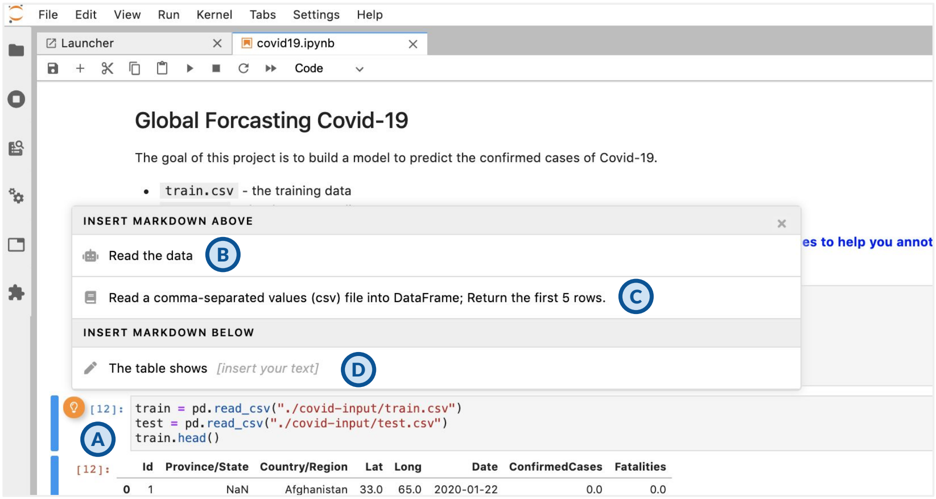

Figure 1: The Themisto user interface is implemented as a Jupyter Notebook plugin: (A) When the recommended documentation is ready, a lightbulb icon shows up to the left of the currently focused code cell. (B – D) shows the three options in the dropdown menu generated by Themisto, (B) A documentation candidate generated for the code with our HAConvGNN model, (C) A documentation candidate retrieved from the online API documentation for the source code, and (D) A prompt message that nudges users to write documentation on a given topic.

The client-side program is responsible for rendering the user interface and monitoring the user actions on the notebook to edits in code cells. When the user's cursor focusing on a code cell, the UI will send the current code cell content to the server-side program through HTTP requests.

The server-side program takes the code content and generates documentation using both the deep-learning-based approach and the query-based approach. For the deep-learning-based approach, the server-side program first tokenizes the code content and generates the AST. It then generates the prediction with the pre-trained model. For the query-based approach, the server-side program matches the curated API calls with the code snippets and returns the pre-collected descriptions. For the prompt-based approach, the server-side program sends different prompts (e.g., for interpreting results or for explaining reason) base on the output type of the code cell.

2.2 User Interface Design

Figure 1 shows the user interface of Themisto as a Jupyter Notebook plugin. Each time the user changes their focus on a code cell, as they may be inspecting or working on the cell, the plugin is triggered. The plugin sends the user-focused code cell's content to the backend. Using this content, the backend generates a code summarization using the model and retrieves a piece of documentation from the API webpage. When such a documentation generation process is done, the generated documentation is sent from the server-side to the frontend, and a light bulb icon appears next to the code cell, indicating that there are recommended markdown cells for the selected code cell (as shown in Figure 1.A).

When a user clicks on the light bulb icon, they can see three options rendered in the dropdown menu: (1) the HAConvGNN approach to generate documentation for source code (Figure 1.B); (2) a query-based approach to retrieve the online API documentation for source code (Figure 1.C); and (3) a user prompt approach to nudge users to write more documentation (Figure 1.D). If the user likes one of these three candidates, they can simply click on one of them, and the selected documentation candidate will be inserted into above the code cell (if it describes why for the code), or below it (if it interprets the result of the code).

3 Three Approaches for Documentation Generation

In this section, we briefly describe the three different approaches for documentation generation.

3.1 HAConvGNN Model Approach

We proposed an attention-based hierarchical ConvGNN component to augment a seq2seq model (HAConvGNN), inspired by GNN architecture in [LeClair *et al.*, 2020]. These GNN models can take both the source code's structure (extracted as AST) and the source code's content as input, in comparison to the traditional sequence-to-sequence model architectures, which only take the source code's content as an input sequence. Because a markdown cell can cover multiple, we augment LeClair et al. [LeClair *et al.*, 2020] by encoding adjacent four code cells under a markdown cell to gain a high-level attention weight.

To build a training dataset, we collect the top 10% of the publicly available notebooks from the top 20 popular Kaggle competitions [1]. Then we remove non-English notebooks and transform the data to follow the data structure in [LeClair *et al.*, 2020]. After data preprocessing, our dataset has 28,625

[1]We checked these challenges' data policy and contacted the Kaggle administrators to make sure our data collection process complies with the Kaggle policy.

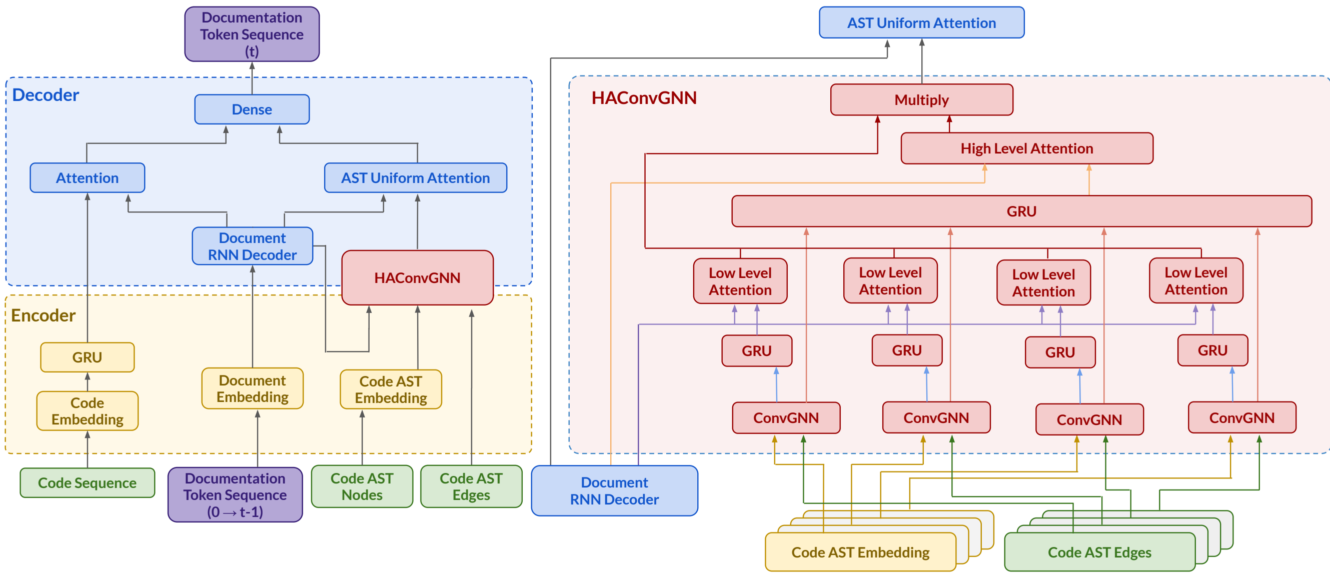

Figure 2: A code summarization model for the deep-learning-based documentation generation approach via GNN.

code(s)-summary pairs. Following the best practice of model training, we split the dataset into training, testing, and validation subsets with an 8 to 1 to 1 ratio. We use the Adamax optimizer [Kingma and Ba, 2014] with a batch size of 20. The learning rate we set is 0.001. The code sequence embedding size is 100. In the encoder, we use GRU [Cho *et al.*, 2014] with the hidden size of 256. The hop size of our GNN is 2. The dropout rate of our attention layer is 0.5.

In our previously published algorithm paper, we reported both quantitative and qualitative evaluations of our model's performance against two baseline models using ROUGE scores [Lin, 2004]. The two baseline models are Code2Seq model [Alon *et al.*, 2018] and Graph2Seq model [Xu *et al.*, 2018] . In this paper, we briefly present the partial quantitative evaluation result. As shown in Table 1, our HAConvGNN model outperforms the other two baselines. For more details, please refer to [Liu *et al.*, 2021].

3.2 Query-Based Approach and Prompt-Based Approach

Well-documented Kaggle notebooks often have the description of frequently-used data science code functions for educational purposes. And sometimes data scientists directly paste in a link or a reference to the external API documentation for a code function [Wang *et al.*, 2021a]. Thus, we implement a query-based approach that curates a list of API from commonly used data science packages, and short descriptions

Model	Precision	Recall	F-Measure
code2seq	11.45	8.46	8.23
graph2seq	13.21	9.87	9.51
HAConvGNN	22.87	16.92	16.58

Table 1: ROUGE-1 scores for baselines and our models

from external documentation sites. In our system, we only cover Pandas , Numpy , and Scikit-learn these three libraries as a starting point to explore this approach. We collected both the API names and the short descriptions by building a crawling script with Python. When users trigger this query-based approach for a code cell, Themisto matches the API names with the code snippets and concatenate all the corresponding descriptions.

Lastly, a well-documented notebook not only documents the process of the code, but also interprets the output, and explains rationales. These types of documentation are hard to generate with automated solutions To achieve it, we implement a prompt-based approach. It detects whether the code cell has a cell output or not: if the cell outputs a result, Themisto assumes that the user is more likely to add interpretation for the output result, thus the corresponding prompt will be inserted below the code cell. Otherwise, the system assumes the user may want to insert a reason or some educational type of documentations, thus it changes its prompt message.

4 Conclusion

In conclusion, this demo paper presents a prototype system that can support Jupyter notebook users to utilize our state-of-the-art NLP algorithm [Liu *et al.*, 2021] to automatically generate documentation for a computational notebook. In the future, we will continue to pursue the intertwined NLP and HCI research agenda [Wang *et al.*, 2021b]: we will design new algorithm models to improve the fully automated documentation generation accuracy, and we will design and evaluate our human-centered AI system's usability with target users.

References

[Alon *et al.*, 2018] Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. code2seq: Generating sequences from structured representations of code. *arXiv preprint arXiv:1808.01400*, 2018.

[Cho *et al.*, 2014] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. 06 2014.

[Kery and Myers, 2017] Mary Beth Kery and Brad A. Myers. Exploring exploratory programming. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 25–29, 2017.

[Kingma and Ba, 2014] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[LeClair and McMillan, 2019] Alexander LeClair and Collin McMillan. Recommendations for datasets for source code summarization. *arXiv preprint arXiv:1904.02660*, 2019.

[LeClair *et al.*, 2020] Alexander LeClair, Sakib Haque, Lingfei Wu, and Collin McMillan. Improved code summarization via a graph neural network. *arXiv preprint arXiv:2004.02843*, 2020.

[Lin, 2004] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[Liu *et al.*, 2021] Xuye Liu, Dakuo Wang, April Wang, and Lingfei Wu. Haconvgnn: Hierarchical attention based convolutional graph neural network for code documentation generation in jupyter notebooks. *arXiv preprint arXiv:2104.01002*, 2021.

[Rule *et al.*, 2018] Adam Rule, Aurélien Tabard, and James D Hollan. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018.

[Wang *et al.*, 2021a] April Yi Wang, Dakuo Wang, Jaimie Drozdal, Xuye Liu, Soya Park, Steve Oney, and Christopher Brooks. What makes a well-documented notebook? a case study of data scientists' documentation practices in kaggle. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–7, 2021.

[Wang *et al.*, 2021b] Dakuo Wang, Pattie Maes, Xiangshi Ren, Ben Shneiderman, Yuanchun Shi, and Qianying Wang. Designing ai to work with or for people? In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–5, 2021.

[Xu *et al.*, 2018] Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, Michael Witbrock, and Vadim Sheinin. Graph2seq: Graph to sequence learning with attention-based neural networks. *arXiv preprint arXiv:1804.00823*, 2018.

[Zhang *et al.*, 2020] Amy X Zhang, Michael Muller, and Dakuo Wang. How do data science workers collaborate? roles, workflows, and tools. *arXiv preprint arXiv:2001.06684*, 2020.