How Data Scientists Improve Generated Code Documentation in Jupyter Notebooks

Michael Muller[a], April Yi Wang[b], Steven I. Ross[c], Justin D. Weisz[d], Mayank Agarwal[e], Kartik Talamadupula[f], Stephanie Houde[g], Fernando Martinez[h], John Richards[i], Jaimie Drozdal[j], Xuye Liu[k], David Piorkowski[l] and Dakuo Wang[m]

[a]*IBM Research AI, Cambridge, MA, USA*
[b]*University of Michigan, Ann Arbor, MI, USA*
The first two authors contributed equally to this paper.
[c]*IBM Research AI, Cambridge, MA 02142 USA*
[d]*IBM Research AI, Yorktown Heights, NY, USA*
[e]*IBM Research AI, Yorktown Heights, NY, USA*
[f]*IBM Research AI, Yorktown Heights, NY, USA*
[g]*IBM Research AI, Cambridge, MA, USA*
[h]*IBM Argentina, La Plata, Argentina*
[i]*IBM Research AI, Yorktown Heights, NY, USA*
[j]*Rensselaer Polytechnic Institute, Troy, NY, USA*
[k]*Rensselaer Polytechnic Institute, Troy, NY, USA*
[l]*IBM Research AI, Yorktown Heights, NY, USA*
[m]*IBM Research AI, Cambridge, MA, USA*

Abstract

Generative AI models are capable of creating high-fidelity outputs, sometimes indistinguishable from what could be produced by human effort. However, some domains possess an objective bar of quality, and the probabilistic nature of generative models suggests that there may be imperfections or flaws in their output. In software engineering, for example, code produced by a generative model may not compile, or it may contain bugs or logical errors. Various models of human-AI interaction, such as mixed-initiative user interfaces, suggest that human effort ought to be applied to a generative model's outputs in order to improve its quality. We report results from a controlled experiment in which data scientists used multiple models – including a GNN-based generative model – to generate and subsequently edit documentation for data science code within Jupyter notebooks. In analyzing their edit-patterns, we discovered various ways that humans made improvements to the generated documentation, and speculate that such edit data could be used to train generative models to not only identify *which* parts of their output might require human attention, but also *how* those parts could be improved.

Keywords

Code-documentation, Generative AI, Human-AI collaboration, Jupyter notebooks

1. Introduction

For several decades, scholars have explored how humans and computers might collaborate [1, 2, 3, 4, 5]. Early work largely focused on a zero-sum "trade-off" model in which a finite conceptual pool of "initiative" was to be split between human and computer. Typical approaches asked, in effect, "who goes first?" and many models went no further than a *single* cycle of human-initiates-and-AI-responds or AI-initiates-and-human-responds (e.g., [6]).

More recent work has deconstructed the older concept of unitary "initiative" into a flexible and collaborative framework in which increased initiative by one party (e.g. human) does not imply a decrease of initiative by the other (e.g. AI) [7]. In addition, the "mixed initiative creative interface" (MICI) framework analyzed by Deterding et al. [1] and Spoto and Oyelnik [5], further developed by Muller et at. [3], specifically details how human and AI partners interact in creative tasks as a *series* of back-and-forth exchanges.

In this paper, we examine how humans interact with a generative AI model in the context of writing data science documentation. We specifically aim to extend

Joint Proceedings of the ACM IUI 2021 Workshops, April 13-17, 2021, College Station, USA
michael1_muller@us.ibm.com (M. Muller); aprilww@umich.edu (A. Y. Wang); slross@us.ibm.com (S. I. Ross); jweisz@us.ibm.com (J. D. Weisz); Mayank.Agarwal@ibm.com (M. Agarwal); krtalamad@us.ibm.com (K. Talamadupula); Stephanie.Houde@ibm.com (S. Houde); martferc@ar.ibm.com (F. Martinez); ajtr@us.ibm.com (J. Richards); drozdj3@rpi.edu (J. Drozdal); liux27@rpi.edu (X. Liu); david.piorkowski@ibm.com (D. Piorkowski); Dakuo.Wang@ibm.com (D. Wang)
0000-0001-7860-163X (M. Muller)

the human-initiates-and-AI-responds interaction pattern to include a step in which a human may make subsequent edits to the outputs of the model. Prior work by our team has explored how data scientists use various kinds of models – including a GNN-based generative model – to aid the task of adding documentation to data science code in Jupyter notebooks [8]. In this paper, we conduct a deeper analysis on the edits made by participants in that study, to understand the nature of their edit-patterns and how they "compensate" or "augment" the output of the generative models. Through a thematic analysis, we developed a classification of participants' edit-patterns.

We found that 85% of participants' edit-patterns fully accepted the algorithmically-generated text, or built upon the generated text; and only 15% of the instances involved a complete rewrite. At first glance, these results suggest that the generated text was well-accepted. However, participants modified the generated text in 41% of the cases. Thus, (1) a human requests the generation of text; (2) the AI provides that text; and then (3) the human makes a decision to use - or not to use - that text, and (4) may go on to edit the generated text. In the future, these three-to-four dialogic "moves" could become the basis for an extended conversation between human AI.

In this paper, we develop a taxonomy of edit-patterns, discovering that some edits added missing details while other edits explained the function of the code. A third category of edits was primarily concerned with modifying the formatting or style of the documentation.

2. Background

We discuss recent work in the area of AI and machine learning applied to data science and software engineering, as well as the application of generative models to this domain. We also discuss recent studies on human interactions with generative models in software engineering.

2.1. AI and Machine Learning in Software Engineering

In recent years, techniques from AI and machine learning have been applied to various tasks in software engineering, including code completion [9, 10, 11, 12], code translation [13], code classification [14, 15], API recommendation [16, 17], variable and method naming [18, 19], type inference [20, 21], bug detection and repair [22, 23, 24, 25, 26, 27], comment description and generation [28, 29, 30, 31, 32, 33], code change summarization [34], and code clone detection [35]. Allamanis et al. [36] provides a comprehensive review of the use of AI and machine learning within data science and software engineering. The emergence of generative AI techniques for natural language, such as GPT-2 [37] and GPT-3 [38], have also been reflected in code-centric use cases: Brockschmidt et al. [39] proposed the use of generative models for source code, and Tufano et al. [40] used generative models to fix bugs. In this paper, we conduct a deeper analysis of participant interactions with Wang et al. [8]'s Themisto documentation-generation system, which incorporates a GNN-based generative model for generating comments from code.

2.2. Human-AI Collaboration with Generative Models in Data Science and Software Engineering

In a recent study, Weisz et al. [41] examined the use of an unsupervised neural machine translation (NMT) model in addressing a task in application modernization, specifically regarding translating code from a legacy language to a modern one. They found that software engineers would be tolerant of errors or mistakes in the output of an NMT model, as the code produced by that model would be subject to the same review and testing procedures as code produced by everyone else on their team. In addition, a code highlighting feature that indicated *where* in the code human attention might be needed, based on an aggregation of the model's token-level confidences, was very desirable. Although a subsequent analysis by Agarwal et al. [42] demonstrated how current metrics of model confidence may not necessarily correlate with external truth of code quality (approximated by lint errors), the ability for a generative model to be able to "ask for help" via code highlights was nonetheless highly valued by software engineers. In this work, we push even further by seeking to understand whether a generative model can also specify *what kind* of it help it needs from its human user.

3. Method

In order to understand the co-creation process of data science documentation, we examined Themisto [8] a prototype code documentation generation system that supports data scientists in writing documentation for computational notebooks.[1] Wang et al. [8] conducted an evaluation of Themisto with 24 data science professionals. In this section, we briefly discuss how Themisto generates documentation from code, as well as the user study setup and data collection methodology.

[1]While Jupyter notebooks have been used in education, we note that these notebooks are increasingly the basis of commercial products [43, 44], as in offerings by IBM [https://developer.ibm.com/components/jupyter/] and Microsoft [https://notebooks.azure.com/].

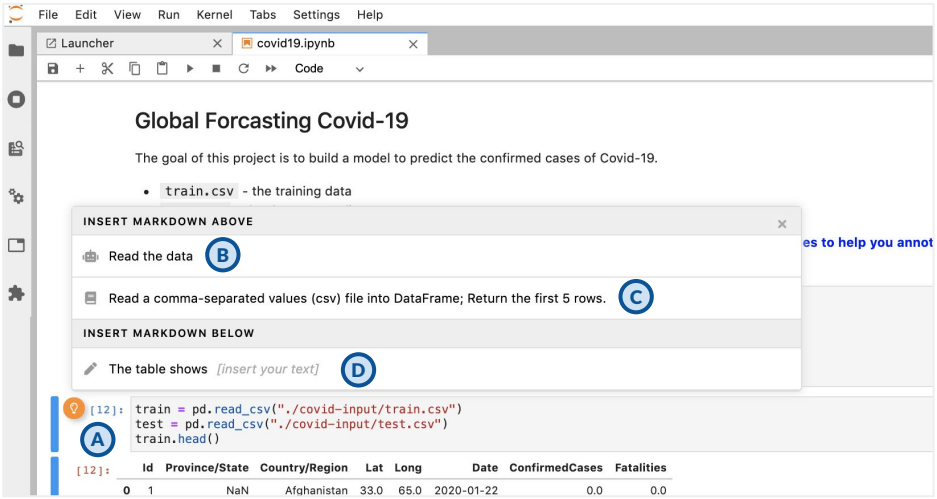

Figure 1: The Themisto user interface [8] is implemented as a Jupyter Notebook plugin: (A) When the recommended documentation is ready, a lightbulb icon shows up to the left of the currently-focused code cell. (B – D) shows the three options in the dropdown menu generated by Themisto, (B) A documentation candidate generated for the code with a deep-learning 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.

3.1. Themisto: A System for Automatic Documentation Generation

We implemented the automatic documentation generation system as an extension to JupyterLab (Figure 1). The extension generates three types of documentation for a given code snippet. The first type of documentation is generated using a Graph Neural Network based approach [45] which is commonly used in code summarization tasks. The second type of documentation is generated by retrieving relevant external API documentation for a code function (e.g., functions defined in Pandas[2], Numpy[3], and Scikit-learn[4]). Lastly, the extension also provides a prompt-based approach where users are given a short prompt to manually create the documentation. For example, if a code cell contains a graphic output, the extension would generate the prompt to ask users to interpret the output.

3.2. User Study Setup

We are interested in how users make revisions on the suggested explanations. Thus, we recruited 26 data scientists to add documentation to a given draft notebook using the prototype. We prepared two draft notebooks with the same length (9 code cells) and similar levels of difficulty, but for two different problems. Each participant was randomly given one of the two notebooks and asked to document the notebook within 12 minutes; two participants failed to complete the task within that time-period, and were excluded from further analysis. Before the study, we provided a quick demo on the functionality of the extension.

3.3. Data Collection

We collected the completed final notebooks (N=24) after participants finished the task. All study sessions were conducted remotely using a teleconferencing tool and we recorded participants' screens with their permissions. After the session, we conducted a retrospective interview to ask about their experience and feedback.

We wanted to understand how participants used the algorithmically-generated documentation. For orientation, we review that 13 participants made documentation choices and optional modifications in each of nine markdown cells in each of two notebooks ("Covid" and "house") - i.e., 13 participants for each notebook. In the data from each notebook, we discovered two participants (per notebook) who did not complete the task, or who created extra cells. We could not be certain how to "map" these extra cells to the structure that was common across the other 11 participants. Because we wanted to compare participants' responses in a disciplined way, we treated each of these people (who created extra cells) as outliers, and excluded them from analysis. This exclusion left us with 11 participants per notebook who had worked with

[2]https://pandas.pydata.org/docs/reference/index.html
[3]https://numpy.org/doc/stable/reference/
[4]https://scikit-learn.org/stable/modules/classes.html

Table 1
Edit-Patterns - Applicable as Modifications to Algorithmically-Generated Text

Source text	Participant text	Participant+Cell
Details - Current		
### Evaluate a score by cross-validation	### evaluate a score by [5-fold] cross-validation [using [rmse]	P15-house+9
### Fill NA/NaN values using the specified method	### [replace] na/nan values [with the mean]	P07-house+6

the algorithmically-generated documentation.

3.3.1. Preparatory Analysis

To prepare the documentation for analysis, we grouped all of the texts for each cell together (separately for each notebook). We then used a bag-of-words method to identify words (tokens) that were *not included* in the algorithmically-generated documentation. In the quotations in this paper, the participant-introduced words appear in bracketed [blue-ink].[5] Table 1 provides an illustrative example. This was a slightly conservative method for identifying new text, because we might fail to detect that a participant had typed "data" (for example) in their own usage, rather than including the word "data" from the algorithmically-generated text. However, we used this method only to orient ourselves to the texts.

We next read each text by each participant. After readings all of the texts, two researchers agreed upon a codebook of edit-patterns. One researcher than applied that codebook rigorously to all of the texts.

3.3.2. Reference Notation

We identify each text in terms of the participant number (1-26, with two non-completers and two outliers excluded), the notebook ("covid" or "house"), and the cell in the notebook (1-9). For example, "p21-house+4" refers to participant 21, in the "house" notebook, in the 4th markdown cell.

4. Results: High-Level Quantitative Comparisons

Participants accepted the algorithmically-generated documentation *unchanged* in 45% of the cells, and they edited the algorithmically-generated documentation in 41% of the cells. The remaining 9% of the cells were left blank. In this workshop paper, we present new analyses to examine the edit-patterns in the 41% of the cells with participant-edited documentation.

There were few statistically significant differences between participant data from the two notebooks. We briefly report those analyses here, before the qualitative analyses that are the core of this paper. Because we did not find differences between the two notebooks, we will then perform content analyses of participants' text on the combined data from the two notebooks in Sections 5 and 6.

4.1. Starting Points for Documentation

We provided three different Sources of documentation: AI, Query, and Prompt. A chi-square analysis found no significant differences in the proportions of Sources chosen by participants in each notebook. We also looked at combinations of Sources - i.e., no discernable source vs. a single source vs. multiple sources. Again, a chi-square analysis showed no significant differences between the notebooks.

4.2. Edit-Patterns

We describe distinct Edit-Patterns below in Sections 5 and 6. Here, we briefly state that we used chi-square tests to examine whether participants used different edit-patterns between the two notebooks. Only one of edit-patterns showed a significant difference, and that pattern was concerned with the format (not the content) of the documentation (i.e., levels of headers in the markdown cell).

Similarly to the "combinations" analysis of Section 4.1, we found no significant differences across notebooks for cells with zero edit-patterns, a single edit-pattern, or multiple edit-patterns.

[5] Readers who use a screenreader may want to consult https://doccenter.freedomscientific.com/doccenter/doccenter/rs25c51746a0cc/2012-06-20_TextFormatting/02_TextFormatting.htm for information on how to access font-attributes through JAWS.

5. Results: Content-Related Edit-Patterns

The preceding quantitative analyses showed only a single, stylistic difference in participants' work with the two notebooks. We therefore combine our qualitative analyses of edit-patterns across the two notebooks.

We manually coded the edit-patterns in each participant's text in each markdown cell, according to our codebook (Section 3.3.1). For the 22 participants in nine markdown cells, we thus coded 99 texts in each notebook, for a total of 198 coded markdown cells. We applied an informal version of thematic analysis [46], noting Braun's and Clarke's advice that there are multiple ways of conducting a thematic analysis [47]. Previous grounded theory and thematic analysis studies have involved from 6 to 74 participants [48, 49], and so our sample of 22 participants is within that conventional range. Within this sample, we used the saturation practices of Guest et al. [50] (recommended by Ando et al. [46]) and Majid et al. [51], defining saturation by a code that appeared from at least two participants. We made no restrictions on the number of codes that we identified in a single text. Thus a text might have zero codes if the participant simply accepted the algorithmically-generated documentation, or it might have as many as three or four different codes in complex cases.

5.1. Details Edit-Patterns (three subcategories)

Participants expanded on the generated documentation by adding details. There were three subcategories of details: *Contextual* information, information about *This-step* (the current step), and information about *Subsequent* steps.

5.1.1. Contextual Details

Contextual details could take several forms. P03-house clarified how the prior steps had produced materials that were used in the current step:

- ### create the target and the test data [re-create training] and test [datasets based on] the [size of] the [original training dataset] (p03-house+7)

By contrast, P210-covid focused on the treatment of missing values

- ### check for [any] missing values [note] that [province/state have quite a few] missing (p021-covid+5)

and P01-covid provided an even-more-detailed account of the same issue, with commentary on what they had observed:

- check for [missing/null] values for [some countries/regions there is no province/state data this is probably correct and not a flaw in] the [data] (p01-covid+5)[6]

5.1.2. This-step

This-step edit-patterns occurred in many distinct subcategories. The first subcategory is the addition of a few words to clarify the current step:

- ### importing libraries ### importing [the necessary] libraries (p20-covid+1)

While P20-covid's addition might be only a matter of emphasis, other simple additions provided much more specific information about what was being done in the step. P08-covid changed the meaning of the generated documentation by adding specificity about *what value* was being computed:

- ### check [number of] the missing values (p08-covid+5)

P03–house provided a different kind of specificity about the types of datasets being used:

- ### read the [training and test datasets] (p03-house+2)

P13-covid engaged in the same type of edit-pattern (in the other notebook), but included much greater detail:

- ### read a comma-separated values (csv) file into dataframe [of training] data [and test] data return the first 5 rows [of] the [training] data (p13-covid+2)

We contrast P08-covid, P03-house, and P13-covid, who were adding information about *what* was being calculated or input, vs. P07-house and P26-house, who described *how* the operations were done:

- ### create [train] and test data [by splitting dataframe] (p07-house+7)
- ### create the target and the test data and [use slicing] (p026-house+7)

The preceding pair of examples suggests that participants may solve the same problem, in the same notebook-cell, in different ways. We found many examples of different strategies and/or different conceptions of what the intended reader would need to know, such as this contrast between P13-covid's rather minimalist description, vs. P01-covid's much more extensive description:

- ### replace a specified phrase [(_)] with another specified phrase [( ) then transform] the [datatype to int] (p13-covid+4)

[6] We note that P01-covid edited-*out* the markdown formatting command, "###". We will have more to say about this kind of stylistic edit-pattern, below.

- ### data [preparation in] the [training] data [set] replace the [dashes] with [spaces for] the [date column and] convert the data [type to] integer (P01-covid+4)

In some cases, participants added specific algorithmic details that none of the generated texts had included. A repeated example was to mention (and sometimes discuss) root mean square error and its importance:

- ### evaluate a score by cross-validation [uses rmse as an evaluation metric] with [5-fold] cross-validation (p03-house+9)
- ### evaluate a score by [5-fold] cross-validation [using rmse] (p15-house+9)

5.1.3. Next-Step

We found a third edit-pattern that anticipated the next step (i.e., a subsequent cell) in the notebook. P08-covid briefly stated the use that the inputted data would be put to:

- ### read [and sanity check] the data (p04-covid+2)

However, in other cases, the participant provided a much richer description of the next steps, as in this recitation by P05-covid:

- ### Model A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting the [first line below initiates] a model [instance] and the [second line] fits the model on the [training data] (p05-covid+8)

5.1.4. Details Patterns Summary

We have shown three edit-patterns in which participants have provided more detail than was available in the generated texts. These patterns might be considered as spanning an imagined audience's reading experience. In some cases, participants wrote Contextual information into the generated texts. This contextual information was generally *retrospective* - i.e., what should the reader have known in order to understand the code? In contrasting cases, participants focused less on context, and more on content within the *current* cell (This-step). Finally, in a few cases, participants wrote to *anticipate* the next cell or cells. In the Discussion, we will think further about this kind of participants' mental model of their audience's experiences.

We also wish to acknowledge that some of our category boundaries are fuzzy. The next category of edit-patterns poses the question - what is the difference between a Details edit-pattern and an Explanation edit-pattern? With further research, we may need to redraw this boundary.

5.2. Explanation Edit-Patterns

Sometimes participants went beyond simple details, writing a more extended Explanation. P02-house and P15-house provided brief examples, in which they added operational explanations of how to perform the activity in the cell:

- ### return the first 5 rows [(defvalue=5)] (p02-house+3)
- ### [separate train] and test [subsets post feature engineering set] the target [as saleprice] (p15-house+7)

P02-covid and P14-house went further, documenting the nature of source data files and their formats, and also the functional significance of additional modules:

- read the [data: from] the [two files: 'traincsv' and 'testcsv' they contain] data [in csv format now 'train' contains] the [train] data [and 'test' contains] the [test] data [start on] the [train] data first (p02-covid+2)
- ### importing libraries - pandas for [dataframes (like excel spreadsheet)] - numpy for [fast vector operations - sklearn] for [simple] data analysis [(in] this [case linear model)] (p14-house+1)

The Explanation edit-pattern brings in different types of information, including operational aspects and extended explanatory material about data files and programmatic resources. As we noted above, with more data, we may discover that Explanations may need to be combined with Details. Another possibility is that Explanations may turn out to be a subset of Tutorial edit-patterns, which we describe in the next subsection.

5.3. Tutorial Edit-Patterns

In a more complex pattern, participants appeared to be teaching the reader how to do the analysis. For example, P05-covid provide detailed explanations about how to carry-out a series of operations in python:

- ### convert [training] data [remove dashes ('-') in] the [dates this is done by applying] the ['replace' function 'astype' sets] the ['date' column to integer type] (p05-covid+4)

Similarly, P03-house gave instructions about how to work with several datasets, including which columns (factors) were involved and how to process those columns:

- [## dataset preparation] the [next few cells prepare] the [train and test datasets] ### concatenate the [train and test datasets] with a [subset of columns (mssubclass to salecondition)] is [format(a b)] (p03-house+4)

In a different cell, P05-covid explained the meaning of a function call and gave further instruction about how to use the results of the function:

- ### check the missing values detect the [number of] missing values for [each column ‘isnull()‘ returns] an [array of indicators of whether each value in a column is] missing [and ‘sum()‘ calculates] the [total number of] missing values [along] that [column] (p05-covid+5)

P11-house taught how a conceptual operation worked and also gave advice about the naming of the statistical action:

- [#####] this code cell is for [handling] missing values [which are replaced with] the [mean value] for [that feature] this is [also known as column-wise mean-imputation] (p11-house+6)

Finally, we note that P24-covid took a somewhat different tutorial strategy. They left the original generated text intact as provided by the algorithm, and then added a link to more information about the python code-structures that were used in the cell that the algorithm had described:

- **p24-covid/expt** ### replace a specified phrase with another specified phrase [[for more information about lambda](https://realpythoncom/python-lambda/)] (p24-covid+4)

Tutorial edit-patterns went much further than Explanation edit-patterns (which themselves had gone further than Details edit-patterns). Tutorial edit-patterns provide not only how-to information, but also interpretations of the meaning or purpose of actions, and in one case a link to further information.

5.4. Rewriting Edit-Patterns

In the preceding subsection, we began to describe a dimension of increasing complexity in the information that participants provided, and also (we infer) increasing effort on the part of the informants to think "beyond" what was given in the generated text. The last edit-pattern in this series involves even more "beyond" work: beyond previous transformations of the generated text, and probably beyond previous levels of effort. We call this edit-pattern "Rewriting," because it involves nearly complete replacement of the generated text.

It may be that the generated text in certain cells led to more instances of Rewriting. For example, three different participants rewrote the generated text of cell 4 of the House notebook:

- ### [join features from train and test into one df] (p15-house+4)
- ### [transform and clean] the [data] (p22-house+4)
- ### [concat train and test col salecondition] (p02-house+4)

Similarly, two people rewrote the contents of cell 9 of the Covid notebook:

- ### [make predictions] (p17-covid+9)
- ### [test] to [see how] the [model performs] (p20-covid+9)

While it initially appears that Rewritten cells were relatively brief, we found that two other participants Rewrote the same cell (cell 9 of the Covid notebook) at much greater length:

- ### [run] the [model] to [generate predictions] on the [test data] and [store them as a ‘dataframe‘] (p04-covid+9)
- use the [trained model] to [predict] the [target] on the [test data] (p05-covid+9)

In some cases, partipants Rewrote the generated text at a higher level of sophsitication:

- ### [one hot encode] the [features] (p15-house+5)

And in one case, the participant Rewrote in a very summary fashion, only listing a series of steps:

- [modeling process - subsetting data - cleaning data - getting rid of nulls -model training] (p14-house+4)

In the Rewriting edit-pattern, we see diverse strategies, ranging from brief summaries to extensive new text, as well as high-level abstractions. Significantly, in multiple cases, participants made distinctly different Rewritings of the same generated text (i.e., in the same cell of the notebook). Thus, while the category of the edit-pattern is the same, the individual strategies can be quite different. We recall that we saw analogous patterns in the This-step Details edit-patterns of subsesction 5.1.2. While there may be agreement among participants that the generated text in certain cells requires a certain type of change, participants clearly adopt different strategies about how to make those changes.

5.5. Content Edit-Patterns Summary

In this part of Results, we have examined how participants changed the contents of the generated texts. Figure 2 summarizes a dimension that runs from simple Details, to more complex Explanations, to instructive Tutorials, and finally to complete Rewritings of the generated text. Collectively, participants have an extensive repertoire of edit-patterns that they apply to particular problems in particular documentation cells. We next examine more stylistic changes that participants applied to generated text.

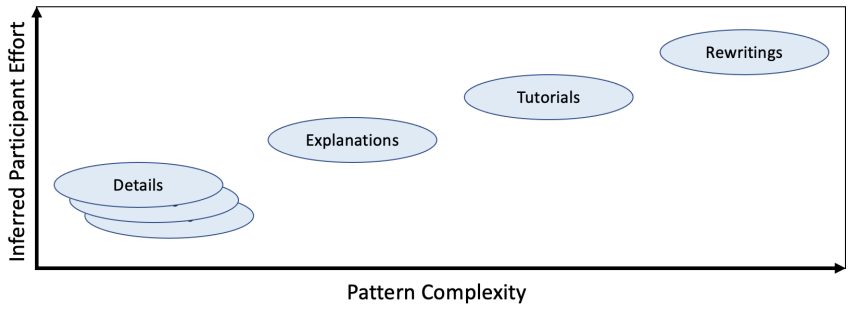

Figure 2: Dimension of Content Edit-Patterns.
Figure description: Graphical display of four Content Edit-Patterns, in a two-dimensional abstract sapce. The horizontal axis represents the complexity of the edit-patterns. The vertical axis represents the inferred amount of participant effort to make the changes. The four classes of edit-patterns run from the lower left (low-low) to the upper right (high-high), in the order of Details, Explanations, Tutorials, and Rewritings.

6. Results: Stylistic Edit-Patterns

6.1. Modifying document hierarchies

Sometimes in combination with other edit-patterns, participants modified the markdown formatting from the generated texts. Initially, all markdown texts were provided at the same hierarchical level (###). In multiple cases, participants modified those levels, placing texts in super-order / sub-order relation to one another:

- [#####] this code cell is for [handling] missing values [which are replaced with] the [mean value] for [that feature] this is [also known as column-wise mean-imputation] (p11-house+6)

P17-covid modified the header markdown specification when combining the contents of two different forms of generated text:

- ### [create] a [classifier ####] random forest [classifier] (p17-covid+8)

Further research will be needed to understand if these stylistic/formatting edits are related to changes to the words in the documentation.

6.2. Completing a Sentence

Some of the changes to content appeared to clarify *what* was being done in the code. The primary subcategory of these changes was to add a verb to a noun-phrase:

- ### [fit] the model (p01-covid+8)
- ### [train the] model (p10-house+8)

Some participants added the verb in a different position in the sentence. In these two examples, we see P21-covid and P24-covid modifying the same generated text, but with different versbs:

- ### model [creating] (p21-covid+8)
- ### model [training] (p24-covid+8)

However, participants also engaged in more complex ways of completing a sentence. For example, P01-covid both added a verb and changed the object of that verb:[7]

- [generate] the [predictions] (p01-covid+9)

While editing the same cell, P02-house, P07-house, and P15-house added the same verb, but then made different modifications to the object of that verb:

- ### [fit regression] model (p02-house+8)
- ### [fit] a lasso linear model [to the training data] (p07-house+8)
- ### [train] lasso [cv] linear model (p15-house+8)

There were also even more complex cases, in which it is not clear if the participant's purpose was to complete a sentence Here, we repeat one example of P17-covid from the previous subsection, which illustrates our point about the ambiguity of complex cases:

- ### [create] a [classifier ####] random forest [classifier] (p17-covid+8)
- ### [leverage] the random forest model and [fit] the model [with training] dataset [(a] random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control [over-fitting)] (p13-covid+8)

[7] We use "verb" and "object" in the technical senses of English-language grammar (e.g., [52]). A "verb" performs an action. An "object" receives the effect of that action.

6.3. Conversational Tone

We observed a further stylistic modification which appeared to make the generated text more conversational. To avoid asserting our own judgments of what "conversational" might mean, we show only examples in which the participant added a personal pronoun - typically "we" or "you". For ease in reading, we have bolded those pronouns in the following examples:

- importing libraries: [in] this code [segment **you** import the python] libraries [first that include 'numpy'] and ['panda' **you** also import] a [class from sklean if **you** need to display some warning import the warnings] library [as shown] (p21-covid+1)
- ### [define] and [configure] the model a random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting [**we** also train] the model [with 'fit()'] (p04-covid+8)
- [##### here **we**] evaluate the [square-root of] the [5-fold cross-validated mean-squared-error of] the [trained] model with the [training set '(x_train y)' (p11-house+9)
- ### [**we** now show] the [predicted values] (p24-covid+9)

6.4. Stylistic Edit-Patterns Summary

We acknowledge that the distinction between content and style is far from clear (e.g., [53]). Therefore, we consider that our current categorization of Content-Related edit-patterns and Stylistic edit-patterns may require revision. With larger datasets, we may for example conclude that Conversational Tone is more related to Tutorial changes, and less related to "style." The same may occur with Completing a Sentence. We will also need to understand better the relationship of header-styles to content in brief documentation text snippets.

7. Results: Summary Statistics of Edit-Patterns

We computed the percentage of the Content-edited cells in which each of the above Content edit-pattern appeared. Details edit-patterns were the most frequent. This may be unsurprising, because these kinds of edit-patterns took relatively little effort (Figure 2). In general, edit-patterns that were most costly of effort (Tutorial, Rewritten) had lower frequencies of occurrence, with Rewritten edit-patterns occurring in fewer than 15% of the edited cells.

Table 2
Percent of All Edit-Patterns in Edited Cells

Edit-Pattern	Percentage of All Edited Cells
Details-Contextual	5.08%
Details-This-step	33.90%
Details-Next-step	5.08
Explanations	11.02%
Tutorial	8.47%
Rewritten	14.41%
Markdown headers	5.08%
Complete sentence	12.71%
Conversational tone	4.24%

Note: A single cell could contain multiple edit-patterns. Therefore, sums of percentages may not be meaningful.

8. Discussion

These results have implications for the design of algorithmic documentation systems. Taken together with our prior work on TransCoder, we can also see emergent ideas about how people can understand and make use of AI outcomes, even in the absence of formal explanatory systems (e.g., Explainable AI, or XAI). Finally, these two projects point us toward important questions in the design of future human-AI collaborative systems.

8.1. Learning from Participants' Improvements

The results we reported on participants' editing patterns lead us to think about a few implications to further improve the automatic documentation approach. First, the Details edit-patterns, Explanation edit-patterns, and Tutorial edit-patterns are relevant to the purpose of the notebook and the target audience. We believe that a future version of the generative approach should tailor the automatic documentation based on the usage scenario. Data scientists can benefit from more candidate documentation where the level of details varied.

With a larger dataset, we could associate these edit-patterns to particular patterns in the algorithmically-generated texts. Based on those associations, we could modify the algorithms to anticipate the kinds of edits that humans have previously made (e.g., [54]). For example, if we can remove the need for Details-related and Conversational-Tone edits, then humans can focus on higher-value editing, such as Tutorials. We may then see emergent categories of even more task-specific and/or domain-oriented edit-patterns, when humans no longer need to put work into less significant edits.

One way to do this, is to include a reinforcement learning component into the algorithm that could learn from users' modifications to the proposed texts. Our current

GNN model relies on the size of the training dataset to ensure the quality of the results. However, data science code snippets are patternless and are of limited use for generating explanations. In the future, we can combine deep reinforcement learning with our current GNN model [55] to improve the performance and generalizability of the results. This can help provide consistent stylistic documentation in terms of the writing style, sentence structure, and level of details.

8.2. Flawed Generative Outcomes can be Useful Outcomes

In our earlier generative documentation paper [8], we learned that people accepted algorithmically-produced documentation in 45% of the cells. In this workshop paper's analysis of the 41% of edited cells, participants retained at least part of the generated text in over 85% of the cells (Details, Explanations, Tutorials). The fact that they chose to do the extra work to Rewrite in only 15% of the cells, is evidence that they mostly chose to work with imperfect text rather than to replace it.

This outcome is consistent with our previous study of code translation, in which engineers reported that they preferred an imperfect translation to no translation at all. While we hope to improve our generative algorithms in both research programs, we also envision future studies in which we will calibrate the quality of the outcomes, to determine the threshold of "poor quality" below which an algorithm should not be deployed. We could then perhaps provide a more "skeletal" outcome, such as an outline of documentation rather than full-text. We could also treat "poor-quality" instances as higher-priority opportunities for algorithmic improvement.

8.3. Deepening Human-AI Collaborations

We now consider each of our research programs (translation and documentation) in terms of published patterns of human-AI collaboration [6, 1, 2, 4, 7, 5]. In both of our studies, the human provides some initial information (source code in both cases), and the generative algorithm responds with a proposed outcome text (target code or documentation, respectively). After that, with minimalist support, the human has to make their own way - e.g., by choosing among alternative translations for sections of the target code, or by manually editing the documentation.

These patterns remain consistent with simple initiative models (e.g., [2, 4]), and fall short of the richer on-going interactions of some of the experimental MICI applications [1, 5], with their potential of AI-augmentation in support of skilled human work. We anticipate that future versions of both projects could move toward longer and richer exchanges between human and AI (e.g., [6]). If the human edits the target code in the TransCoder project, then a secondary AI (e.g., [12]) might assist by generating completion of the human's new code or suggesting additional modifications to the target to remain consistent with the changes. This secondary AI would be "aware" of the original code, and could provide additional type-ahead support, advice, or consultation as needed. Similarly, if the human edits the generated text in the Documentation project, then a secondary AI (e.g., [56]) could provide assistance with Details types of edits, but could also provide language-quality (Stylistic) support for more complex Tutorial accounts, or even narratives (e.g., [57]).

8.4. How Does a Generative AI Model Ask for Help?

Our work highlights an opportunity for enriching human interactions with generative models. At a base level, a generative model takes input (e.g. code) and produces output (e.g. documentation for that code). Agarwal et al. [42] demonstrate how a generative model can produce confidence scores alongside its output, and Weisz et al. [41] show the utility such scores can have in steering human attention toward reviewing portions of the output in which the model has low confidence. In this work, we demonstrate how having an understanding of the nature of human edit-patterns to a generative model's output can enable a generative model to not only identify *where* human attention is needed, but also *how* human effort can be used to improve the quality of its output.

One of the interesting questions will be exactly how to choose among those alternatives - i.e., *when* is type-ahead useful, and *how* should a watchful but respectful AI intervene with advice, and *what dialogic or other communication structures* should be involved in an on-going AI-human consultation [58, 59, 60])? Our experiences with the NMT algorithm in the TransCoder experiment showed that, with a sufficiently broad beam-search [61], we could generate a manageable set of alternative translations, which could be compared using an algorithm like [62] to determine regions of agreement between the translations as well as regions of uncertainty along with the alternatives considered for the uncertain regions. These alternatives could then serve as informal explanations - e.g., *"Q: Why is the output marked as uncertain in this region? A: Because the algorithm considered multiple possible translations at this point, and this is what they were."* The GNN in the Documentation project might also be modified to produce multiple possible texts, with similar explanatory power (*"Q: Why is this documentation marked as uncertain..."*).

If there are multiple, alternative outcomes with no emergent "most-probable" alternative, then this could become an initiation point at which the algorithm de-

tects the need for human assistance. One way to implement this request-for-help is as a "human-in-the-loop" paradigm, in which the human responds to the needs of the algorithm. We also envision situations in which the human may be in the midst of editing code or text, and may ask the algorithm to serve as a text-assistant for the human's on-going editing work. This could become an example of the "AI in the loop" paradigm that we discussed at last year's workshop. In these ways, we could move from the relatively single-process "initiative" models of [2, 4, 7], and toward a more collaborative and on-going series of interactions as in [1, 5].

8.5. Limitations

In order to conduct a controlled study, we sacrificed ecological validity. We asked participants to document *someone else's* notebook. By contrast, the canonical case in Jupyter notebooks is to document one's own code. A future goal should be a more naturalistic practice of documenting *my notebook*.

Paradoxically, for precision of evaluation, we may also need to perform an even *more* controlled study, in which each person receives only one algorithm's text at a time. This approach could help us to assess each algorithmic approach more independently than in the preliminary experiment in this workshop paper.

We also note that our analytic method could be strengthened in future research. Our bag-of-words approach was insensitive to word-order, and we looked only at patterns of *added* words. Future work should also examine patterns of *deleted* words.

Finally, we note the obvious sampling weaknesses. We conducted a relatively small study in a single institution. We hope to examine similar practices in other settings, and with more participants.

9. Conclusion

In this workshop paper, we have addressed topics in human-AI collaboration in data science and software engineering. We reported text analytic results from a study of generative documentation, showing that participants accepted generated text with or without modification in the majority of instances. These results are consistent with our earlier work, in which engineers were enthusiastic about using imperfect NMT-generated translations of software code. Similarly, participants in this study were also quite ready to accept or to work with imperfect GNN-generated texts. We also analyzed the edit-patterns in the generated text, developing categories that suggest future work directions. Finally, going beyond early unidirectional models of "initiative," we sketched promising directions for longer-term, on-going human-AI collaborations.

References

[1] S. Deterding, J. Hook, R. Fiebrink, M. Gillies, J. Gow, M. Akten, G. Smith, A. Liapis, K. Compton, Mixed-initiative creative interfaces, in: Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems, 2017, pp. 628–635.

[2] E. Horvitz, Principles of mixed-initiative user interfaces, in: Proceedings of the SIGCHI conference on Human Factors in Computing Systems, 1999, pp. 159–166.

[3] M. Muller, J. Weisz, W. Geyer, Mixed initiative generative ai interfaces: An analytic framework for generativeai applications (2020).

[4] R. Parasuraman, T. B. Sheridan, C. D. Wickens, A model for types and levels of human interaction with automation, IEEE Transactions on systems, man, and cybernetics-Part A: Systems and Humans 30 (2000) 286–297.

[5] A. Spoto, N. Oyelnik, Library of mixed initiative creative interfaces, "http://mici.codingconduct.cc", 2017. [Online; accessed 21-December-2020].

[6] J. A. Biles, Genjam: Evolution of a jazz improviser, in: Creative evolutionary systems, Elsevier, 2002, pp. 165–187.

[7] B. Shneiderman, Human-centered artificial intelligence: Reliable, safe & trustworthy, International Journal of Human–Computer Interaction 36 (2020) 495–504.

[8] A. Y. Wang, D. Wang, J. Drozdal, M. Muller, S. Park, J. D. Weisz, X. Liu, L. Wu, C. Dugan, Themisto: Towards automated documentation generation in computational notebooks, arXiv preprint arXiv:2102.12592 (2021).

[9] A. Hindle, E. T. Barr, Z. Su, M. Gabel, P. Devanbu, On the naturalness of software, in: 2012 34th International Conference on Software Engineering (ICSE), IEEE, 2012, pp. 837–847.

[10] V. Raychev, M. Vechev, E. Yahav, Code completion with statistical language models, in: Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2014, pp. 419–428.

[11] M. Bruch, M. Monperrus, M. Mezini, Learning from examples to improve code completion systems, in: Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering, 2009, pp. 213–222.

[12] A. Svyatkovskiy, S. K. Deng, S. Fu, N. Sundaresan,

Intellicode compose: Code generation using transformer, arXiv preprint arXiv:2005.08025 (2020).

[13] B. Roziere, M.-A. Lachaux, L. Chanussot, G. Lample, Unsupervised translation of programming languages, Advances in Neural Information Processing Systems 33 (2020).

[14] L. Mou, G. Li, L. Zhang, T. Wang, Z. Jin, Convolutional neural networks over tree structures for programming language processing, in: D. Schuurmans, M. P. Wellman (Eds.), Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA, AAAI Press, 2016, pp. 1287–1293. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/11775.

[15] V. Jayasundara, N. D. Q. Bui, L. Jiang, D. Lo, Treecaps: Tree-structured capsule networks for program source code processing, arXiv preprint arXiv:1910.12306 (2019).

[16] S. Gu, T. Lillicrap, I. Sutskever, S. Levine, Continuous deep q-learning with model-based acceleration, in: International Conference on Machine Learning, 2016, pp. 2829–2838.

[17] N. D. Q. Bui, Y. Yu, L. Jiang, SAR: learning cross-language API mappings with little knowledge, in: M. Dumas, D. Pfahl, S. Apel, A. Russo (Eds.), Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019, ACM, 2019, pp. 796–806. URL: https://doi.org/10.1145/3338906.3338924. doi:10.1145/3338906.3338924.

[18] M. Allamanis, H. Peng, C. Sutton, A convolutional attention network for extreme summarization of source code, in: International conference on machine learning, 2016, pp. 2091–2100.

[19] U. Alon, M. Zilberstein, O. Levy, E. Yahav, code2vec: Learning distributed representations of code, Proceedings of the ACM on Programming Languages 3 (2019) 1–29.

[20] V. J. Hellendoorn, C. Bird, E. T. Barr, M. Allamanis, Deep learning type inference, in: Proceedings of the 2018 26th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering, 2018, pp. 152–162.

[21] J. Wei, M. Goyal, G. Durrett, I. Dillig, Lambdanet: Probabilistic type inference using graph neural networks, arXiv preprint arXiv:2005.02161 (2020).

[22] B. Ray, V. Hellendoorn, S. Godhane, Z. Tu, A. Bacchelli, P. Devanbu, On the "naturalness" of buggy code, in: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), IEEE, 2016, pp. 428–439.

[23] M. Pradel, K. Sen, Deepbugs: A learning approach to name-based bug detection, Proceedings of the ACM on Programming Languages 2 (2018) 1–25.

[24] M. Vasic, A. Kanade, P. Maniatis, D. Bieber, R. Singh, Neural program repair by jointly learning to localize and repair, arXiv preprint arXiv:1904.01720 (2019).

[25] E. Dinella, H. Dai, Z. Li, M. Naik, L. Song, K. Wang, Hoppity: Learning graph transformations to detect and fix bugs in programs, in: International Conference on Learning Representations, 2019.

[26] M. White, M. Tufano, M. Martinez, M. Monperrus, D. Poshyvanyk, Sorting and transforming program repair ingredients via deep learning code similarities, in: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 2019, pp. 479–490.

[27] V. J. Hellendoorn, P. Maniatis, R. Singh, C. Sutton, D. Bieber, Global Relational Models of Source Code, in: International Conference on Learning Representations, 2020.

[28] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, K. Vijay-Shanker, Automatic generation of natural language summaries for java classes, in: 2013 21st International Conference on Program Comprehension (ICPC), IEEE, 2013, pp. 23–32.

[29] S. Iyer, I. Konstas, A. Cheung, L. Zettlemoyer, Summarizing source code using a neural attention model, in: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2016, pp. 2073–2083.

[30] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vásquez, D. Poshyvanyk, R. Oliveto, Automatically assessing code understandability: How far are we?, in: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2017, pp. 417–427.

[31] X. Hu, G. Li, X. Xia, D. Lo, Z. Jin, Deep code comment generation, in: 2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC), IEEE, 2018, pp. 200–20010.

[32] Y. Wan, Z. Zhao, M. Yang, G. Xu, H. Ying, J. Wu, P. S. Yu, Improving automatic source code summarization via deep reinforcement learning, in: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 2018, pp. 397–407.

[33] U. Alon, S. Brody, O. Levy, E. Yahav, code2seq: Generating sequences from structured representations of code, arXiv preprint arXiv:1808.01400 (2018).

[34] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, G. Canfora, Automatic generation of release notes, in: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014, pp. 484–495.

[35] M. White, M. Tufano, C. Vendome, D. Poshyvanyk, Deep learning code fragments for code clone detection, in: 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2016, pp. 87–98.

[36] M. Allamanis, E. T. Barr, P. Devanbu, C. Sutton, A survey of machine learning for big code and naturalness, ACM Computing Surveys (CSUR) 51 (2018) 1–37.

[37] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, Language models are unsupervised multitask learners, OpenAI blog 1 (2019) 9.

[38] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, arXiv preprint arXiv:2005.14165 (2020).

[39] M. Brockschmidt, M. Allamanis, A. L. Gaunt, O. Polozov, Generative code modeling with graphs, arXiv preprint arXiv:1805.08490 (2018).

[40] M. Tufano, C. Watson, G. Bavota, M. D. Penta, M. White, D. Poshyvanyk, An empirical study on learning bug-fixing patches in the wild via neural machine translation, ACM Transactions on Software Engineering and Methodology (TOSEM) 28 (2019) 1–29.

[41] J. D. Weisz, M. Muller, S. Houde, J. Richards, S. L. Ross, F. Martinez, M. Agarwal, K. Talamadupula, Perfection not required? human-ai partnerships in code translation, in: Proceedings of IUI 2021, 2021.

[42] M. Agarwal, K. Talamadupula, S. Houde, F. Martinez, M. Muller, J. Richards, S. Ross, J. D. Weisz, Quality estimation & interpretability for code translation, arXiv preprint arXiv:2012.07581 (2020).

[43] M. B. Kery, M. Radensky, M. Arya, B. E. John, B. A. Myers, The story in the notebook: Exploratory data science using a literate programming tool, in: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, 2018, pp. 1–11.

[44] J. M. Perkel, Why jupyter is data scientists' computational notebook of choice., Nature 563 (2018) 145–147.

[45] A. LeClair, S. Haque, L. Wu, C. McMillan, Improved code summarization via a graph neural network, arXiv preprint arXiv:2004.02843 (2020).

[46] H. Ando, R. Cousins, C. Young, Achieving saturation in thematic analysis: Development and refinement of a codebook, Comprehensive Psychology 3 (2014) 03–CP.

[47] V. Braun, V. Clarke, Using thematic analysis in psychology, Qualitative research in psychology 3 (2006) 77–101.

[48] C. M. Baker, L. R. Milne, R. E. Ladner, Understanding the impact of tvis on technology use and selection by children with visual impairments, in: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, 2019, pp. 1–13.

[49] A. Pradhan, B. Jelen, K. A. Siek, J. Chan, A. Lazar, Understanding older adults' participation in design workshops, in: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, 2020, pp. 1–15.

[50] G. Guest, A. Bunce, L. Johnson, How many interviews are enough? an experiment with data saturation and variability, Field methods 18 (2006) 59–82.

[51] M. A. A. Majid, M. Othman, S. F. Mohamad, S. A. H. Lim, Achieving data saturation: evidence from a qualitative study of job satisfaction, Social and Management Research Journal 15 (2018) 66–77.

[52] B. S. Azar, D. A. Azar, R. S. Koch, Understanding and Using English Grammar: Workbook, Longman, 2000.

[53] G. Lakoff, A figure of thought, Metaphor and symbol 1 (1986) 215–225.

[54] K.-H. Zeng, M. Shoeybi, M.-Y. Liu, Style example-guided text generation using generative adversarial transformers, arXiv preprint arXiv:2003.00674 (2020).

[55] P. Almasan, J. Suárez-Varela, A. Badia-Sampera, K. Rusek, P. Barlet-Ros, A. Cabellos-Aparicio, Deep reinforcement learning meets graph neural networks: exploring a routing optimization use case, arXiv (2019) arXiv–1910.

[56] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, Exploring the limits of transfer learning with a unified text-to-text transformer, arXiv preprint arXiv:1910.10683 (2019).

[57] R. Kazman, G. Abowd, L. Bass, P. Clements, Scenario-based analysis of software architecture, IEEE software 13 (1996) 47–55.

[58] E. Horvitz, Uncertainty, action, and interaction: In pursuit of mixed-initiative computing, Intelligent Systems (1999) 17–20.

[59] S. Ross, E. Brownholtz, R. Armes, Voice user interface principles for a conversational agent, in: Proceedings of the 9th International Conference on Intelligent User Interfaces, 2004, pp. 364–365.

[60] S. Ross, E. Brownholtz, R. Armes, A multiple-application conversational agent, in: Proceedings of the 9th International Conference on Intelligent User Interfaces, 2004, pp. 319–321.

[61] C. Wilt, J. Thayer, W. Ruml, A comparison of greedy search algorithms (2010).

[62] E. Myers, An o(nd) difference algorithm and its variations, Algorithmica 1 (1986) 251–266.