Evaluating Transfer Learning for Simplifying GitHub READMEs

Haoyu Gao
The University of Melbourne
Melbourne, Victoria, Australia
haoyug1@student.unimelb.edu.au

Christoph Treude
The University of Melbourne
Melbourne, Victoria, Australia
christoph.treude@unimelb.edu.au

Mansooreh Zahedi
The University of Melbourne
Melbourne, Victoria, Australia
mansooreh.zahedi@unimelb.edu.au

ABSTRACT

Software documentation captures detailed knowledge about a software product, e.g., code, technologies, and design. It plays an important role in the coordination of development teams and in conveying ideas to various stakeholders. However, software documentation can be hard to comprehend if it is written with jargon and complicated sentence structure. In this study, we explored the potential of text simplification techniques in the domain of software engineering to automatically simplify GitHub README files. We collected software-related pairs of GitHub README files consisting of 14,588 entries, aligned difficult sentences with their simplified counterparts, and trained a Transformer-based model to automatically simplify difficult versions. To mitigate the sparse and noisy nature of the software-related simplification dataset, we applied general text simplification knowledge to this field. Since many general-domain difficult-to-simple Wikipedia document pairs are already publicly available, we explored the potential of transfer learning by first training the model on the Wikipedia data and then fine-tuning it on the README data. Using automated BLEU scores and human evaluation, we compared the performance of different transfer learning schemes and the baseline models without transfer learning. The transfer learning model using the best checkpoint trained on a general topic corpus achieved the best performance of 34.68 BLEU score and statistically significantly higher human annotation scores compared to the rest of the schemes and baselines. We conclude that using transfer learning is a promising direction to circumvent the lack of data and drift style problem in software README files simplification and achieved a better trade-off between simplification and preservation of meaning.

CCS CONCEPTS

• **Software and its engineering → Documentation**; • **Computing methodologies → Neural networks**; • **Applied computing → Text editing**.

KEYWORDS

Software Documentation, GitHub, Text Simplification, Transfer Learning

ESEC/FSE '23, December 3–9, 2023, San Francisco, CA, USA

ACM ISBN 979-8-4007-0327-0/23/12…$15.00
https://doi.org/10.1145/3611643.3616291

ACM Reference Format:
Haoyu Gao, Christoph Treude, and Mansooreh Zahedi. 2023. Evaluating Transfer Learning for Simplifying GitHub READMEs. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23), December 3–9, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3611643.3616291

1 INTRODUCTION

Software documents describe key information about software products, such as technologies, code structure, system design, and architecture. These documents are an integral part of the software development process [30] as they can be used to describe the application requirements, design decisions, architecture, as well as deployment and installation. In particular, README files shape developers' first impression about a software repository and document the software project that the repository hosts [31]. However, README documents often contain jargon, abbreviations, and code blocks, making the text challenging to comprehend for non-specialists and people from other language backgrounds. In fact, readability issues and complicated documents are important issues that practitioners frequently encounter [1]. A simple search on GitHub for "complicated README" yields over 27,000 issues and 37,000 pull requests[1]. Therefore, simplification of README files is needed to improve the efficiency of communication between members of development teams and to propagate new technologies to broader fields.

Significant advancement in Natural Language Processing (NLP) has been witnessed over the last decade, thanks to the development of artificial neural networks. Text Simplification (TS) is an NLP task that focuses on rewriting texts into simpler versions while preserving the original meaning to the extent possible. The simplification of text in the general domain has been studied extensively, and its data sources include mainly Wikipedia and its "Simple-Wikipedia" counterpart [26] as well as Newsela articles written for specific age groups [51]. These data sources, especially their simplified versions, are written by professionals with the intention of catering to people with different levels of reading ability. The simplification of sentences in general domain text can be implemented by three main operations, including splitting, deletion, and paraphrasing [9]. These simplification operations are implicitly encoded in text simplification datasets of the general domain, and neural simplification models trained on these datasets memorise the rules in their parameters and achieve competitive performance [10, 54].

The significant disparity between the simplification of general domain documents and domain-specific software documentation prohibits the simple reuse of text simplification systems designed for the general domain. First, the text style of software documentation includes many code blocks and external links such as URLs, which

[1] https://github.com/search?q=complicated+readme&type=issues

do not resemble the style for general-domain texts. In addition, the simplification rules for software documents differ from the general-domain text by performing fewer deletions and more elaborations. Indeed, in this paper we show that a state-of-the-art approach trained on general-domain documents from Wikipedia is not able to produce semantically identical and/or grammatically correct content in around 80% of the cases when applied to README files (cf. Section 6).

A simplification system trained directly on general-domain text cannot simplify software documentation satisfactorily. To the best of our knowledge, there is no previous research focusing on simplifying software documentation. To address this gap, we collected a software-specific text simplification dataset and trained a simplification system on the data. We experimented with transferring general-domain text simplification rules to software documentation and evaluated the system through automatic metrics and manual analysis. We found that by applying transfer learning, the model was able to generate the most satisfactory simplifications. Specifically, the best-performing model achieved a 34.68 BLEU score in the test set and exceeded the baseline models in terms of semantic similarity, grammar, and simplicity, based on human annotation.

The contributions of our research include the following:

- A README files simplification dataset.
- A pipeline to collect such a dataset.
- An exploration of the application of transfer learning to the problem of simplification of README files with promising results.

2 RELATED WORK

2.1 Documentation Issues and Solutions

Previous studies provide rich empirical evidence for software documentation issues. Steinmacher et al. [43] discovered several barriers to participating in Open Source Software (OSS) projects, one related to "Documentation problems". After that, Aghajani et al. [1, 2] conducted empirical studies to investigate software documentation issues and practitioners' perspectives. Software documentation issues could generally be categorised into what information is contained and how the information is presented.

To address the problems, the automatic generation of software documentation could potentially mitigate correctness, completeness, up-to-dateness and various other issues. Automatic software documentation generation can be applied to various software artifacts, including source code [14, 24, 41], bug reports [34, 35] and pull requests [20]. In terms of source code, Sridhara et al. [41] used algorithms to generate comments for Java methods, while McBurney and McMillan [22] improved it by adding surrounding contexts. Moreno et al. [24] summarised Java classes using stereotype rules and manually defined templates. Hu et al. [14] proposed using a sequence-to-sequence model and formulated code summarisation as a translation task.

In terms of bug reports, Rastkar et al. [34, 35] trained a classifier to identify important sentences from bug reports and used them as summaries. Regarding API documentation, Treude and Robillard [44] augmented API documents using insight sentences from Stack Overflow. Pull request descriptions can also be generated considering commits and code comments [20]. Source code changes are also used to generate commit messages [8, 50].

Automatic documentation generation could help developers identify components that are prone to be overlooked and improve development efficiency. Among the issues, readability is an important issue that practitioners frequently encounter [1]. One of the practitioners in Aghajani et al.'s survey stated, "A developer in our team created confusing and overly complicated documentation for customers of our solution". To the best of our knowledge, no previous studies focused on simplifying software documentation to improve people's comprehension. Text simplification is an NLP technique that could bridge this gap and enhance developers' understanding of software documentation.

2.2 Text Simplification

Multiple data sources have been proposed for the task of simplification of text. Zhu et al. [56] first used Wikipedia and Simple Wikipedia as a source, which was later expanded by Zhang and Lapta [52] to the WikiLarge dataset. However, Xu et al. [51] argued that this Wikipedia-based simplification dataset is suboptimal and difficult to generalise to other genres of text. They proposed a new dataset called Newsela, which contains different levels of simplification. Moreover, there are also simplification corpora of languages other than English [4, 12, 40, 42]. Due to easier access to a large corpus of Wikipedia data, we performed part of our experiment on Wikipedia datasets.

Meanwhile, the success of a text simplification system is highly dependent on the quality and quantity of complex-simple sentence pairs in the training corpus [16]. Zhu et al. [56] first used sentence-level TF-IDF (term frequency inverse document frequency) similarity to construct the alignment between a simple Wikipedia corpus and its regular counterpart. Later, more sophisticated sentence alignment techniques were proposed that consider sentence orders and word-level similarity [6, 15, 17, 48], increasing the alignment quality and the dataset size. Recently, Jiang et al. [16] proposed a neural-based Conditional Random Field (CRF) aligner, which decomposes the potential function into semantic similarity (approximated by the BERT classifier) and alignment label transition (approximated by the feedforward network). Their model automatically aligns 604k non-identical aligned and partially aligned sentence pairs. This powerful tool is able to achieve more than 0.9 F1 score on the previous Wikipedia corpus alignment task, thus making their auto-aligned dataset of higher quality. Sentence alignment is the first procedure in the pipeline of text simplification. In our research, we borrow their idea to align software document pairs as our first step in building a software documentation simplification system.

Recent work began to see text simplification as a monolingual translation task. Specia [40] first applied statistical machine translation to text simplification. Kauchak [18] incorporated regular and simple sentences to train an n-gram language model to perform text simplification tasks. Nisioi et al. [28] began to see text simplification as a task similar to machine translation (MT) and trained a standard sequence-to-sequence model based on LSTM that surpasses the performance of previous statistical MT models. Different network designs were also developed for the model to learn a more effective simplification. Zhang and Lapta [52] used

reinforcement learning for simplification with a reward that approximates simplicity, relevance, and fluency. Zhang et al. [53] combine lexical simplification with sentence-level simplification by first performing lexical substitution and then feeding the sentence into a constrained sequence-to-sequence model. Nishihara et al. [27] proposed a controllable simplification system by adding a level token and modifying the loss function, while Mallinson and Lapata [21] did it employing syntactic and lexical constraints. Current text simplification systems use transformer architecture [16] and can achieve state-of-the-art performance. In our work, we primarily used the transformer model and explored the simplification rule gap between software and general-domain documentation.

When there is a disparity between the data distribution (such as the text styles for software documentation and documents of the general domain), the performance of the model can be degraded [38], in which case transfer learning is needed. Transfer learning improves the performance of a learner in one domain by transferring information from a related domain [47]. It is widely used in many areas, including image processing [13] and natural language processing [7], and has achieved significant success. In our work, we experimented with various transfer learning techniques for the task of software documentation simplification. We applied the knowledge learnt from general-domain document simplification to mitigate the noisy and sparse attributes of software-related texts.

3 DATA COLLECTION

In order to obtain enough software-related documents to train our model, we collected README files from GitHub using its RESTful API. We implemented a program using GitHub access tokens to iterate from the very first repository ordered by GitHub id, and check for candidates for the simplification dataset. We only considered repositories not forking other repositories and with at least ten stars to filter out toy projects. One reason we collected older repositories is that we believe README files in older repositories need more simplification, as different techniques were used back then, and more old repositories have gone through simplification updates compared to recent repositories. As we needed to get updates in the READMEs, longer commit histories will be more likely to contain candidate data. Therefore, only projects with at least 100 commits are investigated. The left part of Figure 2 describes the procedures for collecting the data.

Specifically, for each repository, we iterated through its entire commit history. We collected a list of keywords that can be a hint for simplification. We identified those commits that contain at least one of those keywords and only modify the README file as document simplification instances. The previous README file is marked as the difficult version, and the newly committed file is marked as the simplified version. To encourage more prominent simplifications and avoid training data duplication, we only preserved the first commit and the last commit with simplifications on the README file for each repository. We collected 14,588 document-level regular-to-simple instances in total.

Regarding keyword selection, we initially chose three keywords, i.e., "simplify", "clarify", and "explain". Their definitions were searched in WordNet [23], and their synonyms were further added to the set of keywords. After that, we expanded the keyword set by adding different forms, including nouns, verbs, and adjectives. The keywords, along with their distribution in the final collected corpus, are listed in Table 1. Looking at the table, the keywords "clarify", "simplify", "explain" and "ease" along with their derivations are the most frequently used keywords in the collected documents. The more complicated words like "elucidate" and "comprehend" were rarely used. To provide readers with more information on the effect of these keywords, we also added sample commit messages with the most common word sets and listed them in Table 1.

Table 1: List of Keywords and their Distribution in Data

keywords	count	sum	sample commit message
simplification	51	2,756	Simplify intro paragraph
simplify	1,524		
simple	1,161		
simplicity	20		
reduction	20	314	Change link text to reduce confusion
reduce	294		
clarification	954	7,039	Clarifying README a bit
clarify	3,924		
clear	1,677		
clarity	484		
elucidation	1	2	Elucidate what we do with errorCode.
elucidate	1		
elucidative	0		
elucidatory	0		
explanation	1,412	3,419	Update the documentation to explain how this works
explain	1,983		
explanatory	24		
comprehension	10	14	more comprehensible
comprehend	1		
comprehensible	3		
ease	46	1,044	Rewrote README.md to make it easier to follow
easy	998		

Although keywords in commit message histories convey information about the modified contents, using a unigram of occurrence can be ambiguous. For example, a simple negation term could render the semantics of simplification to the opposite meaning. Also, sometimes the hint word for simplification might not refer to the harvested README file, but to structures in the code blocks. Furthermore, even if it refers to the simplification of the README file, only a few sentences might be simplified, with most parts remaining unchanged. Therefore, further filtering and preprocessing steps are required, which are illustrated in Section 4.

In terms of implementation detail, we used the authors' access tokens, and use PyDriller [39] to mitigate the impact of the GitHub RESTful API rate limit as much as possible. The collected documents are in JSON format, with fields including difficult document, simple document, commit message, language used, and project fork counts. We collected 14,588 of these document pairs in total, which are used to construct our software document simplification dataset.

Instead of focusing on certain programming languages, we want to investigate the overall simplification of README files through the commit history and thus did not filter on the programming language field. Figure 1 lists the top ten languages used by the repositories that we collected.

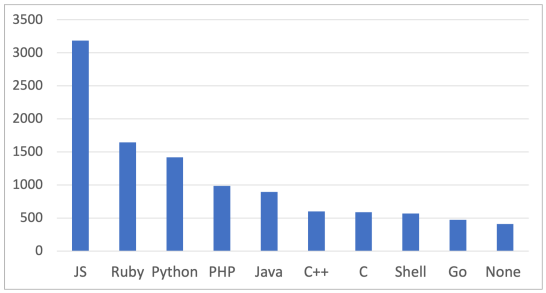


Figure 1: Repository Languages

4 DATA PREPROCESSING

After harvesting GitHub README files, the difficult-to-simple correspondence is at the document level. This is too long to build an effective translation model directly, as most sentences in two versions of documents are duplicates, which creates difficulty for the model to learn simplification. Also, considering that it is not reliable to only depend on heuristic keywords as an indicative sign of simplification, we need to further filter the harvested dataset and perform the sentence alignment task in order to give a higher confidence dataset in a sentence-level correspondence. Figure 2 depicts the overall procedure, with the left hand side describing the dataset construction process. Each component will be discussed in detail in this section.

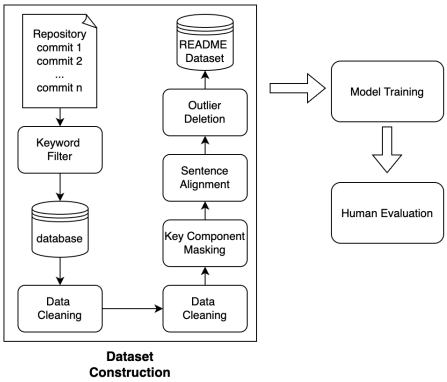


Figure 2: Overall Procedure

4.1 Data Cleansing and Masking

The collected README files are written in several formats, including recommended markdown style[2], plain text and HTML syntax, which makes them noisy. For example, some people follow the conventions "-" and "|" for constructing tables, while others choose to use HTML syntax. For data cleaning and preprocessing, we first removed emojis and different spacing characters including "\t" and "\n".

Table 2: Semantic Components and assigned Tokens

Component Type	Token
inline code block	$\langle code\_small \rangle$
chunks of code	$\langle code\_large \rangle$
path of file or directory	$\langle file \rangle$
table	$\langle table \rangle$
hyperlink	$\langle url \rangle$

Another critical characteristic of our data source is that it contains a large number of semantic components that depend on the document and its context. These components include URLs, code blocks, tables, etc. These components are essential for software documentation, as they usually include instructions, specifications, and external links that elaborate on projects. But it can be challenging for the translation model to implicitly learn their attributes as different documents typically contain components that are barely the same. Therefore, we used Python package "markdown2"[3] to identify and convert these special components into different individual tokens that are distinguishable for their usage. The translation model is explicitly told where the special components are and can generate more cohesive sentences. We manually inspect the data and categorise the tokens into five types. The special tokens are listed in Table 2.

Version requirements and plain text code without using the markdown syntax are also important semantic information in the sentences. However, detecting these elements would require using regular expressions, which is noisy and not the major goal of this paper. Therefore, we leave these elements as in their original form.

4.2 Sentence Alignment

Sentence alignment methods were extensively studied in previous research. Jiang et al. [16] recently developed a neural-based CRF sentence alignment method that achieves an F1 score over 0.9 on Wikipedia data. They decompose the potential function as follows: $\psi(a_i, a_{i-1}, S, C) = sim(s_i, c_{a_i}) + T(a_i, a_{i-1})$, where S denotes simple sentences, C denotes complex sentences and a_i denotes the index of the aligned sentence. A fine-tuned BERT model is used to approximate $sim(s_i, c_{a_i})$, and a simple multi-layer perceptron is used to approximate $T(a_i, a_{i-1})$. They finally used a Viterbi algorithm for decoding the optimal alignment arrangements.

For Wikipedia data, since original and simple documents are not composed concurrently, the positions of difficult sentences and their simplified correspondence could differ a lot. However, the aligned sentences tend to be in a relatively similar order in terms of our harvested GitHub README files due to the incremental development nature of many software projects. Therefore, the calculation of $T(a_i, a_{i-1})$ in our software documents will not benefit much and will only increase training and decoding time. Therefore,

[2] https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/quickstart-for-writing-on-github

[3] https://github.com/trentm/python-markdown2

we discarded other components and only borrowed the fine-tuned BERT classifier to perform our alignment task.

The specific alignment task is performed as follows. For each pair of regular-simple documents, the sentences of the simplified document will be fed into the BERT classifier with the regular sentences one by one. For those that are classified as "aligned", we would temporarily mark them as aligning candidates. To avoid $O(n^2)$ time complexity when performing the alignment task, we exploited the fact that many GitHub README files tend to grow incrementally. Unless a complete refactor of the documents, aligning sentences should appear at the a closer section compared to non-aligning ones. Each sentence in the simplified document will only be compared with the regular ones that have the sentence position within a window size of 50 to the simplified sentence. This window size is able to cover most of the README file sentences, except for excessively long ones, thus reducing the processing time while providing good coverage for the majority of sentence pairs. However, the drifted sentence style for software documentation and the large amount of potential matches for each simple sentence make the false positive rate relatively high. To accommodate this situation, multiple filtering methods rules are used.

First, we filtered the candidates using the TF-IDF-based cosine distance. TF-IDF is a commonly used statistic in natural language processing, which computes weights for each occurring word by taking into account the frequency of the word as well as the frequency of documents containing it. In this case, it does not give great weight to frequently occurring but meaningless words such as "the" and "a". We trained our TF-IDF model on the corpus of all README files. Using the TF-IDF vectorizer, each sentence is represented as a vector, and we are able to compute the cosine distances between simple and complicated sentences.

To filter out false positive candidate pairs using the cosine distance based on TF-IDF, we manually selected 60 simple to complicated sentence pairs and labelled them with the ground truth alignments, with 30 pairs labelled "aligned" and the other 30 as "not aligned". We experimented with different filtering thresholds for cosine distances, and the result is shown in Figure 3.

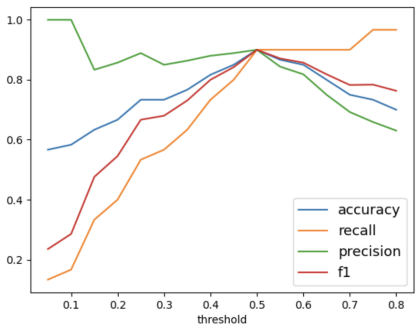


Figure 3: Performance of Different Thresholds

As seen in the figure, the F1 score and recall increase until the cosine distance threshold of 0.5. After that, the recall flattens while both the precision and F1 score start to decrease. As in the real collected alignment candidate pairs, more of them tend to be false positive instead of only taking half the proportion, we choose the threshold of 0.5 to prevent a further drop in the accuracy score. All candidate pairs with TF-IDF cosine distances greater than 0.5 are categorised as false positives and discarded in this step.

Furthermore, the BLEU score [29] is a widely used metric for machine translation tasks that computes the n-gram overlaps between the target and reference sentences and provides an intuitive score for the level of similarity between sentences. A BLEU score greater than 0.9 typically indicates merely a copy of the source sentence, while a BLEU score below 0.1 means overlap only in some name entities [16]. Therefore, we discarded sentence pairs with BLEU scores greater than 0.9 or less than 0.1. The size of the dataset after applying the TF-IDF distance and BLEU score filter is 43,772.

4.3 Dataset Anomaly Filter

After performing the previous alignment steps, we have constructed a dataset of regular-to-simple software documents. To obtain a high-quality dataset, we collected statistics on the number of alignments of regular sentences for each simple sentence and eliminated those that appeared to be outliers. The average alignment number for the simplified sentence is 1.2, with a maximum number of 40 and a variance of 0.8. We then removed data that are outside the range of 3σ. As a result, only sentences that are aligned with no more than three sentences were preserved. We also discarded excessively long sentences. Sentences with more than 40 alphabetic words were eliminated. This procedure further reduces the size of the dataset to 34,667.

After a closer look at these eliminated sentence pairs that were initially categorised as "aligned" by our sentence aligner, most of these outliers either appear to be too short or repeat instructions that only change a few words or URLs. For example, an original document of "$[video](\langle url \rangle)](\langle url \rangle)$", which is a markdown syntax to show some URLs, is matched with three other texts of "$[\langle url \rangle](\langle url \rangle)](\langle url \rangle)$" in the simplified document. These masked sentences are short but similar only in markdown syntax structure instead of semantic meanings, and should be considered as noise. Using this method, we further cleaned up our proposed dataset.

4.4 Dataset Comparison

For the simplicity of elaboration, we refer to the dataset constructed using the Wikipedia and Simple Wikipedia source as "wiki-data" and refer to the dataset we constructed in the previous steps as "sw-data". In this section, we briefly discuss the attributes of both datasets. Table 3 lists statistics for "wiki-data" and "sw-data".

As seen from the table, these two datasets differ significantly. Specifically, sentences in wiki-data tend to have a shorter length. An average length of over 24.80 and 26.62 for simple and regular sentences in the sw-data indicates that sentences harvested from GitHub can be more complex and wordy. Moreover, the vocabulary size in wiki-data significantly surpasses that of the sw-data. This can happen because software documentation only focuses on specific topics, while wiki-data covers a much wider range of topics.

Also, the simple-to-regular ratio statistics in wiki-data and sw-data indicate that the simplification in wiki-data is more aggressive. In contrast, the simplification of sw-data makes less apparent changes. This could happen because Simple Wikipedia articles are

Table 3: Statistics for sw-data

	Simple	Regular	Simple-Regular Ratio
sw-data statistics			
Average Length	24.80	26.62	93.2%
Vocabulary Size	21,653	22,313	97.0%
Exclusive Vocab Size	2,889	3,549	81.4%
wiki-data statistics			
Average Length	14.76	20.91	70.6%
Vocabulary Size	32,228	37,278	86.5%
Exclusive Vocab Size	1,171	6,221	18.8%

written with the intention of letting non-native speakers feel confident in reading. At the same time, the simplification in GitHub files includes different operations such as rewrite, exemplify and clarify, and some of the detail changes are minor. As the sw-data dataset contains relatively less apparent simplification, the model tends to memorise the original sentence and barely performs simplification. The simplification rule gap between wiki-data and sw-data, plus the noise in the sw-data, motivates us to explore transfer learning, as discussed in the next section.

To further illustrate our points, we picked two representative simplification examples, one from wiki-data and the other from sw-data, as can be seen in Table 4. In this example, the wiki-data simplification performs aggressively and ignores some details of the evolution of the presidency armies. However, the author who simplified the sw-data document only changes a few words at the end of the sentence, making the argument clearer by giving a specific instruction.

As we are going to use both datasets to train our software documentation simplification system, we split both datasets into train, validation, and test sets. For the sw-data, we have a train set of size 28,000, a validation set of size 3,500 and the rest of the data forms the test set. For the wiki-data, we have a train set of size 450,000 as well as a validation set and a test set both of size 20,000. The training of the model and the transfer learning will be conducted on the train set, and the performance of cross-entropy loss will be evaluated on the validation set. We will finally generate new texts on the test set for more detailed evaluation.

5 MODEL TRAINING AND TRANSFER LEARNING

In this section, we elaborate on how we trained our model using transfer learning, and discuss the output of the model based on BLEU score evaluation.

5.1 Model Tokeniser

Before feeding sentences into our model, we need to tokenise sentences into a list of tokens so that the model can learn their representations in the embedding layer. The tokens can be whole words or subwords. A tokeniser off-the-shelf is able to perform well on general-domain tasks like simplifying wiki-data. However, software documentation has a lower lexical complexity and contains components that the model does not want to reduce. To better fit our study, a custom tokeniser is needed. Therefore, we trained our own tokeniser using all sentences in the sw-data and wiki-data training set using the WordPiece tokenisation algorithm [49]. WordPiece is a subword tokenisation algorithm developed by Google which is widely used in various models [7, 37]. Similar to Byte Pair Encoding (BPE) [11], it learns how to merge characters into words when provided with a large corpus. During tokenising, a sub-word with a "##" symbol indicates it is the continuation of the previous subword and is later concatenated.

In Section 4.1, we used regular expressions to mask these special components to prevent key components from automatic simplification to different tokens. These special tokens are listed in Table 2. However, to ensure that our tokeniser does not further split these tokens, we specify those as special tokens during our training of the tokeniser, along with $\langle sos \rangle$, $\langle eos \rangle$ and $\langle UNK \rangle$, indicating the start of a sentence, the end of a sentence and unknown words, respectively. In this case, the tokeniser can directly tokenise these components as a whole. As a result, the downstream model will directly know the meaning of these tokens, making it easier for the model to learn how to manipulate them. We also specified the vocabulary size of the tokeniser at 40,000.

5.2 Model Architecture and Hyperparamters

The text simplification task can be considered a translation task, in which sequence-to-sequence models are widely used. Transformer [45] is a multi-headed self-attention sequence-to-sequence model that achieves competitive performance in neural translation tasks. This architecture has become an essential building block in many models in the deep learning area. As this work focuses more on the simplification rules of software documentation and mitigating the drawbacks in the currently collected sw-data, we adopted the vanilla version of Transformer in the original paper [45] with only some minor changes in the tokeniser, and a reduction of trainable parameters to save training time. With limited access to GPU computing resources, plus the long training time of our model, we did not try to tune the hyperparameters extensively to reach the best performance. Instead, we experimented with only a few sets of hyperparameters close to the setting from [16] and used one set of them that performs the best on the task of sw-data. This set of hyperparameters was later used on every model we trained, including the wiki-data and the transfer learning. Specifically, the model configuration and hyperparameter choices are listed in Table 5.

5.3 Training on wiki-data and sw-data

As a starting point, models with the given architecture were trained solely on the wiki-data sets and the sw-data set. The cross-entropy loss curve of the entropy of the model trained with wiki-data is shown in Figure 4. and Figure 5 respectively.

We trained our models on wiki-data for 40 epochs and sw-data for 50 epochs. Their learning curves exhibit similar patterns representing the model gradually overfitted on the training set. In the final epochs, as the loss on the validation set is not decreasing, we stop the training process and preserve checkpoints with the lowest validation error as final models.

Table 4: Simplification Examples

	regular	simple
sw-data	## limitations * due to the nature of irssi's readline, it is not possible to add formatting directly in the input line, hence the need for the extra window kludge.	## limitations * due to the nature of irssi's readline, it is not possible to add formatting directly in the input line, so an extra line is output to the screen instead.
wiki-data	The presidency armies were the armies of the three presidencies of the East India Company's rule in India, later the forces of the British Crown in India, composed primarily of Indian sepoys.	The presidency armies were the armies of the three presidencies of British India.

Table 5: Model Configuration and Hyperparameters

Multi-head numbers	8	Learning rate	1e-5
Encoder layers	6	Batch size	8
Decoder layers	6	Optimiser	Adam
Embedding dimension	512	Alpha	2e-5
Feed-forward dimension	2,048	Dropout rate	0.1

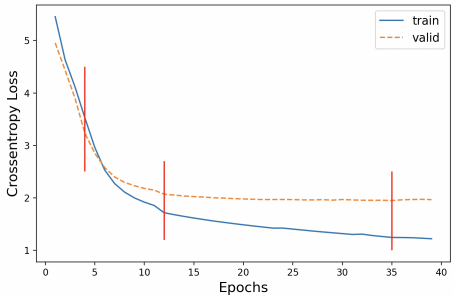


Figure 4: Wiki-data loss curve

5.4 Transfer Learning

Because of the limitations mentioned in the Dataset Comparison section, it is difficult for the wiki-data-trained model to adapt to the change in text style and in simplification rule. Meanwhile, it is difficult for the sw-data-trained model to generate ideal simplifications, as the dataset contains different styles of simplification and many simplifications with only a few swaps of words. These two models are used as baselines to compare our later transferred learning models. For simplicity of argument, we denoted these two models as baseline wiki and baseline sw. A combination of both styles in wiki-data and sw-data, namely practical simplification and technical precision, is desired for software documentation simplification. By applying transfer learning, we intend to share general-domain text simplification knowledge with the software documentation simplification task.

Figure 4 shows three vertical red lines corresponding to the model checkpoint after training for 3 epochs, 12 epochs, and 37 epochs. For simplicity of elaboration, we call them *checkpoint early*, *checkpoint mid*, and *checkpoint best*. In terms of the *checkpoint early*, the model is still under-fitted after only seeing the dataset a few times. Some high-level knowledge for general-domain text simplification has been learnt, but not enough to perform well. With respect to the *checkpoint mid*, the model has established a firm understanding of the text simplification task in the general domain. Also, it is at the "elbow point" for the validation loss curve, meaning that the learning speed decreased significantly after this point. In terms of the *checkpoint best*, the model has overfitted the training set, but its performance on the validation set is the best. We also incorporated the optimiser into the checkpoint for a smoother optimising process.

We adopted different transfer learning paradigms to explore the effect of transfer learning in the software documentation simplification task. Specifically, we started from the *checkpoint early*, mid and best, and used these pre-trained checkpoints to train our models on the sw-data. Figure 5 contains the validation loss curves for all three different transfer learning paradigms and their comparison to the performance of the baseline sw.

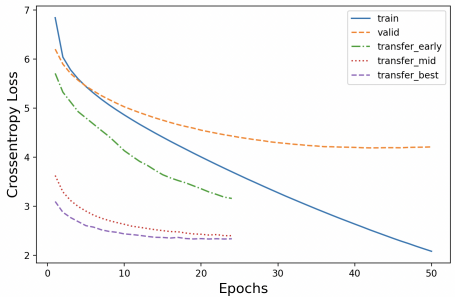


Figure 5: Transfer Learning Loss Curves

As seen in the figure, the cross-entropy loss curves for the three transfer learning strategies all have lower starting points. Moreover, the loss drops faster than the model trained solely on the sw-data. 24 epochs were trained on the three models and their loss in the validation set has reached the minimum. In terms of cross-entropy loss for the three transfer learning models, the *checkpoint best* is the lowest, while the *checkpoint early* is the highest.

However, loss in the validation set is merely an indicator of model performance. This metric suggests the uncertainty level of the model when decoding encoded sentences into their simplifications. The lower this metric, the more confident the model will be. However, as our sw-data dataset contains different writing styles and mask tokens, including URLs and code blocks, the model can find it difficult to generate more fluent sentences. Therefore, better performance in terms of the cross-entropy loss could happen just because the model learnt how to generate more fluent sentences

from the checkpoint of the wiki-data. In this case, we need to look at the model performance in more detail.

5.5 BLEU score Evaluation

Sequence-to-sequence models have an exposure bias problem [33]. Therefore, we use the beam search method, which keeps track of the top k most probable candidate words during the data generation part. We choose the beam size k to be 5. 24 epochs were trained for the three transfer learning models. We also took snapshots of the models after every four epochs of training. For example, for the *checkpoint early* model, we took the checkpoints after it was trained on sw-data for epochs of 4, 8, 12, 16, 20 and 24. We generate simplification text on two baselines, plus all the transfer learning model checkpoints on the test set. The generated texts will be investigated more thoroughly in later sections.

We used the BLEU score to evaluate the quality of the texts generated in the last section. BLEU score measures the similarity of the generated text with its references and is widely used in machine translation tasks. It calculates the n-gram overlaps between the generated text and references. In our experiment, an equal weight of one quarter is given to unigram, bigram, trigram, and 4-gram to compute the BLEU score. The BLEU scores for the baseline sw and the baseline wiki are 13.35 and 14.93. Figure 6 shows the BLEU scores for all other transfer learning models. We also included an off-the-shelf simplification model from Nisioi et al [28], which reached a BLEU score of 25.70.

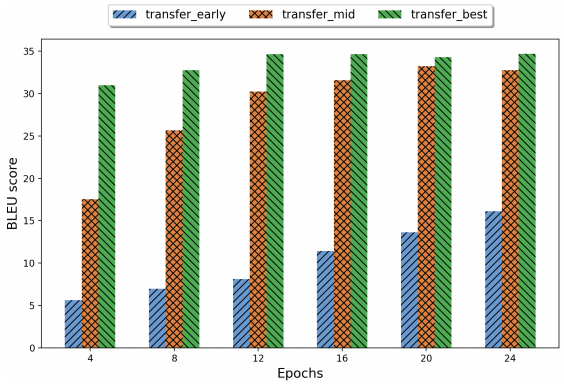


Figure 6: BLEU scores for Transfer Learning Models

As seen from the figure, the BLEU score for transfer_early model starts at the lowest, and it increases gradually to 16.06 after training for 24 epochs, surpassing the BLEU score for both baseline wiki and baseline sw. For transfer_mid model, the BLEU score increases significantly as the training on sw-data proceeds. It eventually reaches a BLEU score of 32.73. On the other hand, the transfer_best performance starts with the highest BLEU score and barely increases as the training proceeds. However, it still has the highest BLEU score of 34.68 in the test set. The BLEU score performance for the transfer_mid model is slightly lower than that of the transfer_best, except for the lower initial points. Furthermore, the BLEU scores for all three models are better than those for the two baselines.

Summary of transfer learning: The baseline model directly trained on sw-data suffered from overfitting. By continuing training the checkpoints for wiki-data, models achieve a lower cross-entropy loss. Meanwhile, the transfer learning models surpass our two baseline models in BLEU score performance. The baseline wiki and baseline sw achieved BLEU scores of 14.93 and 13.35, while our best model achieves a BLEU score of 34.68.

6 HUMAN ANNOTATION

To further verify the effectiveness of the transfer learning approaches, We elaborate on the process of evaluation below.

6.1 Procedure

We selected four models to be evaluated in this phase, including three baseline models and one transfer learning model. The baseline models are the wiki-data trained model, sw-data trained model and the model from Nisioi et al. [28], and we refer to as "Baseline 1", "Baseline 2" and "Baseline 3". The transfer learning model is further trained on the transfer_best checkpoint using sw-data for another 24 epochs, which we refer to as "Transfer".

We adopted a similar method to that of Liu et al. [20] by first randomly selecting 100 original sentences from the sw-data test set, and generating texts using the four models. For each original sentence, there will be four versions of the simplification. For each of the 100 groups, we randomly shuffled the order of these versions to prevent annotators from discovering the patterns and making biased judgements. Also, if models generate the same new texts, they will be reduced to one piece of text for annotators to mark. This approach is used to avoid accidentally giving different marks to the same output.

We conducted a Prolific survey by dividing the 100 questions into 10 different surveys. Each survey was taken by three different participants. We performed a sequential survey release strategy. Specifically, for subsequent survey publications, individuals who had previously participated were excluded from the sample. In this case, participants cannot participate in the study multiple times. Meanwhile, we followed a study design similar to Nadi and Treude [25] by inserting a "quality gate" in a random position in the survey. We used the sentence "The purple monkey dishwasher sang shenanigans on the moon with unicorns and marshmallow socks." consistently in all surveys as the "quality gate". This sentence has semantics clearly irrelevant to the reference sentence. We filtered out participants who did not give a semantic score below three for this sentence.

We used a similar evaluation metric as in related work [**?**]. Annotators are provided with the source sentences and their different simplified versions. During their annotation, they were asked to assess each generated sentence based on three evaluation criteria:

- Simplicity: if the generated sentence is simpler than the original sentence.
- Semantic Similarity: if the generated sentence retains all semantics of the original sentence.
- Grammar: if the generated sentence is grammatically correct.

Likert scale is used to mark each of the aspects, with a score of 1 for strongly disagreeing and a score of 5 for strongly agreeing.

6.2 Demographics of Annotators

We first applied predefined filters from Prolific, which requires the participants to be in the employment sector of Information Technology (IT), while also being proficient in English. We asked the same question about employment again at the beginning of our survey and filtered out 45 participants who did not answer this question consistent with what they registered on Prolific. Our "quality gate" question further filtered out two participants.

In the survey, we asked them about their job roles and how many years they have worked in the IT field. For the 30 participants who passed all the filters and submitted their responses, they have on average 4.4 years of experience working in IT, with a maximum of twelve years and a minimum of half a year. Among the 30 participants, there are ten developers (33.3%), seven IT support staff (23.3%), four project managers (13.3%), four data analysts (13.3%), four administrators (13.3%) and one search quality rater (3.3%).

We computed their level of agreement on the annotation results, which reached 0.42 for Krippendorff's alpha [19] coefficient overall, with semantics, grammar and simplicity at 0.53, 0.37 and 0.32, respectively. This indicates that annotators reached some agreement, while some of the results are subjective at the same time.

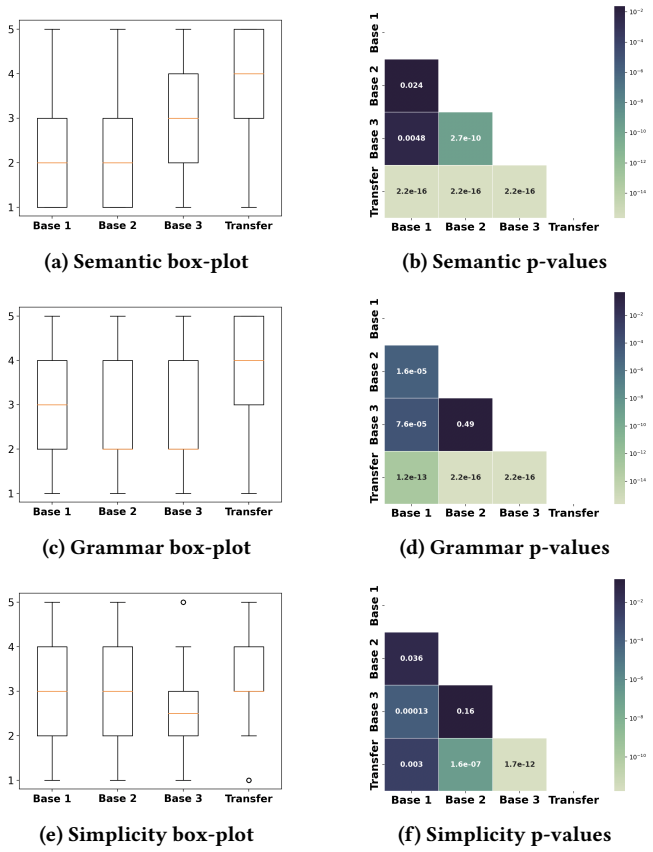
6.3 Results

A good simplification of a sentence should not only be "simpler" than the original sentence, but should also preserve semantics and be grammatically correct. We provide one counter-example, where the original sentence is "Note: to create a debug build of the building files, pass the $--debug$(or $-d$) switch when running the either configure or build command" and the simplified sentence by Baseline 1 is "To create a compact build of the unlimited file, pass the award tells the story of the game". In this example, two annotators gave a semantic score of one, but a simplicity score of four and five, respectively. However, because of the subjectivity of the annotation process and the diversity of the participants, the last annotator gave a score of two in simplicity for this sentence. This sentence does not preserve any meaning from the original sentence. We argue that these generated sentences that either fail to preserve the meaning or are grammatically wrong are not usable. Therefore, we define a "good" sentence as one with a semantic score and a grammar score of at least four.

Table 6 shows the average Likert scores for all models in the three metrics mentioned above, as well as the number of instances with semantic score, grammar score or both no less than four. The left column in Figure 7 shows the box plot for the distribution of these three aspects. We brief the Baseline 1 to three with the name "Base" 1 to 3 to clearly present them in the figure.

Overall speaking, Baseline 2 performs the worst in the semantic and grammar aspects, and second to last for simplicity aspect. Although Baseline 1 is able to generate grammarly moderately acceptable sentence, because it is not trained on sw-data, it fails to generate the domain-specific text, causing the loss of semantic. The simplicity score for Baseline 1 is regarded at an acceptable scale. In terms of Baseline 3, it performs not ideal on all three aspects.

For Transfer model, its scores largely surpass the rest of the models. It is capable of generating more grammatically correct sentences that preserve semantic meanings. Also, it has an average



(a) Semantic box-plot (b) Semantic p-values

(c) Grammar box-plot (d) Grammar p-values

(e) Simplicity box-plot (f) Simplicity p-values

Figure 7: Score Box-plot and Pairwise Wilcoxon p-values

simplicity score of 3.35, indicating that the generated sentences are on average simpler than the original ones.

We can have a more detailed comparison via the box plots in Figure 7. Baseline 3 performs better in terms of semantic preserving, while Transfer model consistently performs better in all three aspects. In terms of the simplicity aspect, Baseline 1 and Baseline 2 occasionally generate satisfying simplification that is above a score of three, while Baseline 3 consistently performs the worst.

From a different perspective, the Transfer model can generate 219 sentences that well preserve the original sentence semantics and 211 sentences that are grammarly correct out of the total 300 sentences according to the survey result. Overall, more than 60% of the sentence generated by the Transfer model is "good", while most of the sentences generated by other models are barely usable.

Additionally, we performed statistical tests to verify whether the differences shown above are statistically significant. We performed Wilcoxon signed rank tests [36] on all pairs of available data on semantic, grammar, and simplicity scores. The right column in Figure 7 includes the heatmaps for all pairs of models, with the calculated p-values listed in cells. One important note is that pairs that are identical to the original sentence (22 out of 100) were not considered when performing the statistical test for simplicity score.

If taking a significance level of 0.05, we can see that the score for Transfer model is statistically better than all other models in

Table 6: Likert score for Models

		Semantics	Grammar	Simplicity	#Semantics≥4	#Grammar≥4	#Good
1	Baseline 1	2.427	3.097	3.017	71	136	54 (18.0%)
2	Baseline 2	2.113	2.637	2.790	45	90	33 (11.0%)
3	Baseline 3	2.730	2.700	2.557	108	98	64 (21.3%)
4	Transfer	**3.907**	**3.873**	**3.350**	**219**	**211**	**182 (60.7%)**

Table 7: Generation Examples

	Original	Simplified
Example 1	If you're interested in using speaker notes, reveal.js comes with a node server that allows you to deliver your presentation in one browser while viewing speaker notes in another.	reveal.js comes with a speaker notes plugin which can be used to present per-slide notes in a separate browser. **(by Transfer model)**
Example 2	gldispatch/ contains code for libgldispatch, which is responsible for dispatching opengl functions to the correct vendor library.	gldispatch/ contains code for libgldispatch. **(by Transfer model)**
Example 3	When a collection is typed as Seq[String], so might have linear access like List, but actually is a WrappedArray[String] that can be efficiently parallelized, but can be efficient with scala parallel collections.	When a collection is typed as Seq[String], so might have linear access like List, but actually is a WrappedArray[String] that can be efficient.(**by Transfer model**)

every aspect. Meanwhile, the differences in grammar score and simplicity score between Baseline 1 and Baseline 2 are statistically significant. This indicates that by applying transfer learning on the wiki-data checkpoint, the model could learn how to generate more fluent sentences that are grammarly correct and perform meaningful simplification.

To further explain these results, we analysed interesting cases in more detail. We found that baselines 1 and 2 have an interesting distribution in terms of simplicity. Although they have relatively lower scores of 2.790 and 3.017, around 25% sentences are higher than four for both Baseline1 and Baseline 2. We further divided the annotation source sentences into one group containing at least two masked special components and another with less than two. For Baseline 1, this results in an average simplicity score of 3.18 for sentences with less than two masked tokens, and the average score drops to 2.74 when there are at least two masked tokens. Meanwhile, the performance for Baseline 2 increases from 2.63 to 3.04 for these two groups.

The decrease in performance for the model trained on wiki-data is reasonable as there are no masked components in general-topic text, and the domain transition would introduce a performance drop. In contrast, the model trained on sw-data sees an increase in performance when more masked special tokens appear in the sentences. Meanwhile, Baseline 1 performs better in generating grammatically correct sentences. Therefore, we argue that wiki-data checkpoint brings more coherent sentences, while further training on sw-data enhances the preservation of semantics and gives the model better versatility with the masked special tokens.

Lastly, we provide three examples of the simplification generated by the Transfer model in Table 7. The first example omits unimportant details while making the sentence easier to understand. The second example discards the second half of the sentence. The third example omits a technical part of "WrappedArray" by not mentioning its parallelization. By rewriting and omitting parts of the sentences in a way that does not severely interfere with the semantics, the Transfer model can provide sentences perceived as simpler by our annotators. The simplicity scores for these three examples were 4.67, 4 and 3.67, on average, while their semantic scores ranged between 4 to 5, 3 to 4 and 2 to 4, respectively.

Summary of human annotation results: Our best model (Transfer) consistently outperforms three baselines in all three aspects. Wiki-data checkpoint enhances the coherence and grammar of the generated sentences, while further training on sw-data improves the preservation of semantics and gives the model better versatility with the masked special tokens.

6.4 Analysis on Identical Sentences

As the sw-data simplification is less prominent than wiki-data, models sometimes learn to predict sentences identical to the original ones. In the 100 sentences used for our annotation, we found that transfer learning models with higher scores tend to generate more replications. Specifically, for the 100 cases, Baseline 2 did not generate any replication, while Baseline 1 and Baseline 3 generated 2 replication each. On the other hand, Transfer generated 22 replications. For example, for the original text "The chain method takes one argument: m.chain(f), f must be a function which returns a value if f is not a function, the behaviour of chain is unspecified.", Transfer model generated identical output, while Baseline 1 generated a shorter sentence with "The variable method takes one argument is a function which returns a value if a mathematical is not a function". The Baseline 1 generated sentence omits many

details and barely retains semantic information. This level of simplification is not practical for developers as details are missing and semantics are degraded. However, it is hard for the model to learn to simplify effectively in each scenario, especially when the sw-data contains more replications. This motivates us to incorporate domain-specific rules in our future work.

7 THREATS TO VALIDITY

We consider threats to the validity of our study in this section.

The first threat is that we mined software repositories from the first GitHub index and did not collect repositories created after 2017. We believe that more README files in older repositories need to be simplified, as different techniques were used back then, and more old repositories have gone through simplification updates compared to recent ones. Collecting datasets from different creation periods of time could potentially give different simplification results.

Second, additional context-related components, such as package requirements, could potentially be masked when assigning new tokens. However, these components are usually embedded in plain text and it would require sophisticated regular expression tools for extracting them. Regular expressions are known to be noisy when processing plain text, and this is not our main purpose in this paper. Therefore, a better regular expression tool to preprocess the text could yield different results on the simplification dataset.

Third, due to the lack of computing resources, we did not extensively tune the hyperparameters on the models, which could lead to overfitting and suboptimal solutions. However, given that the performance gap in the same set of hyperparameters among models is quite obvious, the overall design would not be compromised.

Lastly, we acknowledge that the BLEU score is not an ideal indicator for simplification tasks, and this motivates us to perform a survey with human participants. Although our annotation results revealed that users find that the transfer learning model generates the most satisfying result, our study does not provide evidence on the impact of simplification on comprehension tasks.

8 IMPLICATIONS AND FUTURE WORK

The simplification of README files has significant implications from several perspectives. First, from a newcomer's perspective to a repository, a simpler version of the README files has the potential to help newcomers understand the project structure faster and mitigate the technical barriers. Second, from the perspective of repository owners, an easy-to-read README file could enhance the repository's potential to attract more users and participants. Third, from the perspective of document writers, the recommended simplified version of their text could help them to take care of certain groups of readers when composing the draft.

In addition, as README files usually take the role of project walkthrough and tutorial, similar ideas of simplification could be applied to software teaching materials and other relevant versions of tutorials. This work explores the simplification operations from README documentation and fills the gap between general-style text simplification and domain-specific simplification. Our transfer learning approach provides a direction for using general-style simplification knowledge to compensate for the lack of knowledge in domain-specific simplification settings.

Although we performed a Prolific survey of people with IT backgrounds, one limitation of is that we did not collect longitudinal evidence on the effects of simplified documentation on different stakeholders in open source. Different people interpret "simple" differently, which is indicated by the moderate Krippendorff's alpha score from the Prolific survey. In addition, documents from different programming languages may need different simplifications because of the techniques they use and the communities they are in.

Moreover, Wikipedia data is found to be biased in culture, gender and other perspectives [5, 46]. Although we did not investigate this issue, using the transfer learning model trained on this data, biased use of words might be carried forward. This could be detrimental, especially when some communities are found to be more toxic in language [32]. Therefore, more investigation into this issue is an important direction for future work.

In terms of future work, people with different levels of expertise may find different levels of detail easier. For example, entry-level developers might find a comprehensive document easier to understand. At the same time, people with more expertise might need just enough documents that are "to the point". This situation also applies to different job roles and ages. Therefore, we intend to perform more user-centred studies to elucidate how different groups perceive the concept of "simplification".

In addition, more empirical studies on how README files are updated for readability and simplicity purposes are also in the future direction, specifically: (1) What repositories tend to include more simplification operations? (2) What aspects are the simplification operations focusing on? (3) What triggers the simplification operation? Through qualitative and quantitative studies, we could summarise the gap between people's perceptions and common practices for READMEs simplification, providing more concrete advice on the aspects to pay attention to during documentation writing.

Lastly, to automate the process of human-centred software document simplification, higher-quality data, different metrics, and simplification rules could all be incorporated into the system. Also, in the era of Large Language Models (LLM) [3], we could consider prompt engineering LLM [55] for performing this task.

9 CONCLUSION

In this paper, we collected README files from GitHub and used the BERT sentence alignment algorithm and multiple heuristic filters to construct a README files simplification dataset. Then we trained a transformer model on both wiki-data and sw-data and performed transfer learning by continuing training the model on sw-data from the wiki-data trained checkpoints. After that, we performed a Prolific survey, asking people with IT background to annotate 100 groups of sentences generated by different models from the perspective of semantic preserving, grammarly correctness and simplicity. The best transfer learning mode outperforms the baselines in both the automatic evaluation of BLEU score and the human evaluation. We found that the transfer learning model learns to perform meaningful simplification behaviours to the sentences while preserve the original meaning of the sentences.

10 DATA AVAILABILITY

The replication package is at https://zenodo.org/record/8265001.

REFERENCES

[1] Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C. Shepherd. 2020. Software Documentation: The Practitioners' Perspective. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 590–601. https://doi.org/10.1145/3377811.3380405

[2] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. 2019. Software documentation issues unveiled. In *Proceedings of the 41st International Conference on Software Engineering*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, Montreal, QC, Canada, 1199–1210. https://doi.org/10.1109/ICSE.2019.00122

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[4] Dominique Brunato, Felice Dell'Orletta, Giulia Venturi, and Simonetta Montemagni. 2015. Design and Annotation of the First Italian Corpus for Text Simplification. In *Proceedings of the 9th Linguistic Annotation Workshop*. Association for Computational Linguistics, Denver, Colorado, USA, 31–41. https://doi.org/10.3115/v1/W15-1604

[5] Ewa S Callahan and Susan C Herring. 2011. Cultural bias in Wikipedia content on famous persons. *Journal of the American society for information science and technology* 62, 10 (2011), 1899–1915.

[6] Will Coster and David Kauchak. 2011. Learning to Simplify Sentences Using Wikipedia. In *Proceedings of the Workshop on Monolingual Text-To-Text Generation*. Association for Computational Linguistics, Portland, Oregon, 1–9. https://aclanthology.org/W11-1601

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805 http://arxiv.org/abs/1810.04805

[8] Jinhao Dong, Yiling Lou, Qihao Zhu, Zeyu Sun, Zhilin Li, Wenjie Zhang, and Dan Hao. 2022. FIRA: Fine-Grained Graph-Based Code Change Representation for Automated Commit Message Generation. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE*. ACM, Pittsburgh, PA, USA, 970–981. https://doi.org/10.1145/3510003.3510069

[9] Lijun Feng. 2008. Text simplification: A survey. *The City University of New York, Technical Report* (2008).

[10] Yang Feng, Shiyue Zhang, Andi Zhang, Dong Wang, and Andrew Abel. 2017. Memory-augmented Neural Machine Translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, Martha Palmer, Rebecca Hwa, and Sebastian Riedel (Eds.). Association for Computational Linguistics, 1390–1399. https://doi.org/10.18653/v1/d17-1146

[11] Philip Gage. 1994. A new algorithm for data compression. *C Users Journal* 12, 2 (1994), 23–38.

[12] Núria Gala, Anaïs Tack, Ludivine Javourey-Drevet, Thomas François, and Johannes C. Ziegler. 2020. Alector: A Parallel Corpus of Simplified French Texts with Alignments of Misreadings by Poor and Dyslexic Readers. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*. European Language Resources Association, Marseille, France, 1353–1361. https://aclanthology.org/2020.lrec-1.169

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. https://doi.org/10.48550/ARXIV.1512.03385

[14] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep code comment generation. In *Proceedings of the 26th Conference on Program Comprehension, ICPC*, Foutse Khomh, Chanchal K. Roy, and Janet Siegmund (Eds.). ACM, Gothenburg, Sweden, 200–210. https://doi.org/10.1145/3196321.3196334

[15] William Hwang, Hannaneh Hajishirzi, Mari Ostendorf, and Wei Wu. 2015. Aligning Sentences from Standard Wikipedia to Simple Wikipedia. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Rada Mihalcea, Joyce Yue Chai, and Anoop Sarkar (Eds.). The Association for Computational Linguistics, Denver, Colorado, USA, 211–217. https://doi.org/10.3115/v1/n15-1022

[16] Chao Jiang, Mounica Maddela, Wuwei Lan, Yang Zhong, and Wei Xu. 2020. Neural CRF Model for Sentence Alignment in Text Simplification. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, Online, 7943–7960. https://doi.org/10.18653/v1/2020.acl-main.709

[17] Tomoyuki Kajiwara and Mamoru Komachi. 2016. Building a Monolingual Parallel Corpus for Text Simplification Using Sentence Similarity Based on Alignment between Word Embeddings. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. The COLING 2016 Organizing Committee, Osaka, Japan, 1147–1158. https://aclanthology.org/C16-1109

[18] David Kauchak. 2013. Improving Text Simplification Language Modeling Using Unsimplified Text Data. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL*. The Association for Computer Linguistics, Sofia, Bulgaria, 1537–1546. https://aclanthology.org/P13-1151/

[19] Klaus Krippendorff. 2004. Reliability in content analysis: Some common misconceptions and recommendations. *Human communication research* 30, 3 (2004), 411–433.

[20] Zhongxin Liu, Xin Xia, Christoph Treude, David Lo, and Shanping Li. 2019. Automatic Generation of Pull Request Descriptions. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE*. IEEE, San Diego, CA, USA, 176–188. https://doi.org/10.1109/ASE.2019.00026

[21] Jonathan Mallinson and Mirella Lapata. 2019. Controllable Sentence Simplification: Employing Syntactic and Lexical Constraints. *CoRR* abs/1910.04387 (2019). arXiv:1910.04387 http://arxiv.org/abs/1910.04387

[22] Paul W. McBurney and Collin McMillan. 2014. Automatic documentation generation via source code summarization of method context. In *22nd International Conference on Program Comprehension, ICPC*, Chanchal K. Roy, Andrew Begel, and Leon Moonen (Eds.). ACM, Hyderabad, India, 279–290. https://doi.org/10.1145/2597008.2597149

[23] George A Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J Miller. 1990. Introduction to WordNet: An on-line lexical database. *International journal of lexicography* 3, 4 (1990), 235–244.

[24] Laura Moreno, Jairo Aponte, Giriprasad Sridhara, Andrian Marcus, Lori L. Pollock, and K. Vijay-Shanker. 2013. Automatic generation of natural language summaries for Java classes. In *IEEE 21st International Conference on Program Comprehension, ICPC*. IEEE Computer Society, San Francisco, CA, USA, 23–32. https://doi.org/10.1109/ICPC.2013.6613830

[25] Sarah Nadi and Christoph Treude. 2020. Essential Sentences for Navigating Stack Overflow Answers. In *27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER*, Kostas Kontogiannis, Foutse Khomh, Alexander Chatzigeorgiou, Marios-Eleftherios Fokaefs, and Minghui Zhou (Eds.). IEEE, London, ON, Canada, 229–239. https://doi.org/10.1109/SANER48275.2020.9054828

[26] Courtney Napoles and Mark Dredze. 2010. Learning Simple Wikipedia: A Cogitation in Ascertaining Abecedarian Language. In *Proceedings of the NAACL HLT 2010 Workshop on Computational Linguistics and Writing: Writing Processes and Authoring Aids* (Los Angeles, California). Association for Computational Linguistics, USA, 42–50.

[27] Daiki Nishihara, Tomoyuki Kajiwara, and Yuki Arase. 2019. Controllable Text Simplification with Lexical Constraint Loss. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL Volume 2: Student Research Workshop*, Fernando Alva-Manchego, Eunsol Choi, and Daniel Khashabi (Eds.). Association for Computational Linguistics, Florence, Italy, 260–266. https://doi.org/10.18653/v1/p19-2036

[28] Sergiu Nisioi, Sanja Stajner, Simone Paolo Ponzetto, and Liviu P. Dinu. 2017. Exploring Neural Text Simplification Models. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL*, Regina Barzilay and Min-Yen Kan (Eds.). Association for Computational Linguistics, Vancouver, Canada, 85–91. https://doi.org/10.18653/v1/P17-2014

[29] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. ACL, Philadelphia, PA, USA, 311–318. https://doi.org/10.3115/1073083.1073135

[30] David Lorge Parnas. 2009. Document based rational software development. *Knowledge-Based Systems* 22, 3 (2009), 132–141.

[31] Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. 2019. Categorizing the content of github readme files. *Empirical Software Engineering* 24 (2019), 1296–1327.

[32] Naveen Raman, Minxuan Cao, Yulia Tsvetkov, Christian Kästner, and Bogdan Vasilescu. 2020. Stress and burnout in open source: toward finding, understanding, and mitigating unhealthy interactions. In *ICSE-NIER 2020: 42nd International Conference on Software Engineering, New Ideas and Emerging Results*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, Seoul, South Korea, 57–60. https://doi.org/10.1145/3377816.3381732

[33] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. Sequence Level Training with Recurrent Neural Networks. In *4th International Conference on Learning Representations, ICLR 2016*, Yoshua Bengio and Yann LeCun (Eds.). San Juan, Puerto Rico. http://arxiv.org/abs/1511.06732

[34] Sarah Rastkar, Gail C. Murphy, and Gabriel Murray. 2010. Summarizing software artifacts: a case study of bug reports. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE*, Jeff Kramer, Judith Bishop, Premkumar T. Devanbu, and Sebastián Uchitel (Eds.). ACM, Cape Town, South Africa, 505–514. https://doi.org/10.1145/1806799.1806872

[35] Sarah Rastkar, Gail C Murphy, and Gabriel Murray. 2014. Automatic summarization of bug reports. *IEEE Transactions on Software Engineering* 40, 4 (2014), 366–380.

[36] Denise Rey and Markus Neuhäuser. 2011. Wilcoxon-signed-rank test. In *International encyclopedia of statistical science*. Springer, 1658–1659.

[37] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR* abs/1910.01108 (2019). arXiv:1910.01108 http://arxiv.org/abs/1910.01108

[38] Hidetoshi Shimodaira. 2000. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference* 90, 2 (2000), 227–244.

[39] Davide Spadini, Maurício Finavaro Aniche, and Alberto Bacchelli. 2018. PyDriller: Python framework for mining software repositories. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018*, Gary T. Leavens, Alessandro Garcia, and Corina S. Pasareanu (Eds.). ACM, Lake Buena Vista, FL, USA, 908–911. https://doi.org/10.1145/3236024.3264598

[40] Lucia Specia. 2010. Translating from Complex to Simplified Sentences. In *Computational Processing of the Portuguese Language, 9th International Conference, PROPOR (Lecture Notes in Computer Science, Vol. 6001)*, Thiago Alexandre Salgueiro Pardo, António Branco, Aldebaro Klautau, Renata Vieira, and Vera Lúcia Strube de Lima (Eds.). Springer, Porto Alegre, RS, Brazil, 30–39. https://doi.org/10.1007/978-3-642-12320-7_5

[41] Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori L. Pollock, and K. Vijay-Shanker. 2010. Towards automatically generating summary comments for Java methods. In *ASE 2010, 25th IEEE/ACM International Conference on Automated Software Engineering*, Charles Pecheur, Jamie Andrews, and Elisabetta Di Nitto (Eds.). ACM, Antwerp, Belgium,, 43–52. https://doi.org/10.1145/1858996.1859006

[42] Sanja Stajner, Iacer Calixto, and Horacio Saggion. 2015. Automatic Text Simplification for Spanish: Comparative Evaluation of Various Simplification Strategies. In *Recent Advances in Natural Language Processing, RANLP*, Galia Angelova, Kalina Bontcheva, and Ruslan Mitkov (Eds.). RANLP 2015 Organising Committee / ACL, Hissar, Bulgaria, 618–626. https://aclanthology.org/R15-1080/

[43] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David F. Redmiles. 2015. Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing, CSCW*, Dan Cosley, Andrea Forte, Luigina Ciolfi, and David McDonald (Eds.). ACM, Vancouver, BC, Canada, 1379–1392. https://doi.org/10.1145/2675133.2675215

[44] Christoph Treude and Martin P. Robillard. 2016. Augmenting API documentation with insights from stack overflow. In *Proceedings of the 38th International Conference on Software Engineering, ICSE*, Laura K. Dillon, Willem Visser, and Laurie A. Williams (Eds.). ACM, Austin, TX, USA, 392–403. https://doi.org/10.1145/2884781.2884800

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. https://doi.org/10.48550/ARXIV.1706.03762

[46] Claudia Wagner, David García, Mohsen Jadidi, and Markus Strohmaier. 2015. It's a Man's Wikipedia? Assessing Gender Inequality in an Online Encyclopedia. In *Proceedings of the Ninth International Conference on Web and Social Media, ICWSM*, Meeyoung Cha, Cecilia Mascolo, and Christian Sandvig (Eds.). AAAI Press, University of Oxford, Oxford, UK, 454–463. http://www.aaai.org/ocs/index.php/ICWSM/ICWSM15/paper/view/10585

[47] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. 2016. A survey of transfer learning. *Journal of Big data* 3, 1 (2016), 1–40.

[48] Kristian Woodsend and Mirella Lapata. 2011. Learning to Simplify Sentences with Quasi-Synchronous Grammar and Integer Programming. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Edinburgh, Scotland, UK., 409–420. https://aclanthology.org/D11-1038

[49] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR* abs/1609.08144 (2016). arXiv:1609.08144 http://arxiv.org/abs/1609.08144

[50] Shengbin Xu, Yuan Yao, Feng Xu, Tianxiao Gu, Hanghang Tong, and Jian Lu. 2019. Commit Message Generation for Source Code Changes. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI*, Sarit Kraus (Ed.). ijcai.org, Macao, China, 3975–3981. https://doi.org/10.24963/ijcai.2019/552

[51] Wei Xu, Chris Callison-Burch, and Courtney Napoles. 2015. Problems in current text simplification research: New data can help. *Transactions of the Association for Computational Linguistics* 3 (2015), 283–297.

[52] Xingxing Zhang and Mirella Lapata. 2017. Sentence Simplification with Deep Reinforcement Learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017*, Martha Palmer, Rebecca Hwa, and Sebastian Riedel (Eds.). Association for Computational Linguistics, Copenhagen, Denmark, 584–594. https://doi.org/10.18653/v1/d17-1062

[53] Yaoyuan Zhang, Zhenxu Ye, Yansong Feng, Dongyan Zhao, and Rui Yan. 2017. A Constrained Sequence-to-Sequence Neural Model for Sentence Simplification. *CoRR* abs/1704.02312 (2017). arXiv:1704.02312 http://arxiv.org/abs/1704.02312

[54] Sanqiang Zhao, Rui Meng, Daqing He, Andi Saptono, and Bambang Parmanto. 2018. Integrating Transformer and Paraphrase Rules for Sentence Simplification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, Brussels, Belgium, 3164–3173. https://doi.org/10.18653/v1/d18-1355

[55] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large Language Models are Human-Level Prompt Engineers. In *The Eleventh International Conference on Learning Representations, ICLR*. OpenReview.net, Kigali, Rwanda. https://openreview.net/pdf?id=92gvk82DE-

[56] Zhemin Zhu, Delphine Bernhard, and Iryna Gurevych. 2010. A Monolingual Tree-based Translation Model for Sentence Simplification. In *COLING 2010, 23rd International Conference on Computational Linguistics, Proceedings of the Conference*, Chu-Ren Huang and Dan Jurafsky (Eds.). Tsinghua University Press, Beijing, China, 1353–1361. https://aclanthology.org/C10-1152/

Received 2023-02-02; accepted 2023-07-27