frontiers in Computer Science

ORIGINAL RESEARCH
published: 05 August 2021
doi: 10.3389/fcomp.2021.628832

Readability Enhancement for PDF Documents

Chen-Hsiang Yu, Zachary Shelton, Omar Abou Nassif Mourad and Mohamed A. Oulal*

Wentworth Institute of Technology, School of Computing and Data Science, Boston, MA, United States

Readability has been studied for decades, ranging from traditional paper reading to digital document reading, Web page reading, etc. Different audiences have different needs and the needs trigger the researchers to investigate innovative solutions. For example, in recent years, researchers have studied readability enhancement of English articles for non-native English readers, either on paper reading or hypertext document reading. Using a variety of methods, researchers were able to enhance the reading comprehension and the users' satisfaction on hypertext document reading, such as changing content presentation with visual-syntactic text formatting (VSTF) format or *Jenga* format. In terms of dynamically changing content presentation for reading, one less explored format is Portable Document Format (PDF), which was traditionally viewed within a modern Web browser or Adobe Acrobat reader on the desktop. PDF format was standardized as an open format in 2008 and has been widely used to keep a fixed-layout content. However, a fixed layout document presents a challenge to apply existing transformation methods, not mention on mobile devices. In this paper, we not only present a system that uses a novel algorithm to decode PDF documents and apply content transformation to enhance its readability, but we also generalize it to a framework that allows the users to apply customizations and the developers to customize their needs. Although we used *Jenga* format as an example to enhance the readability of PDF documents, we envision the proposed framework can be used to adopt different customizations and transformation methods. The current result is promising, and we believe it is worth further investigation to make PDF documents readable and accessible for different populations, such as non-native English readers, people with dyslexia or special needs, etc.

Keywords: readability enhancement, PDF documents, mobile devices, content transformation, accessibility

OPEN ACCESS

Edited by:
Carlos Duarte,
University of Lisbon, Portugal

Reviewed by:
Patricia Acosta-Vargas,
University of the Americas, Ecuador
Rosa Navarrete,
Escuela Politécnica Nacional, Ecuador

****Correspondence:***
Chen-Hsiang Yu
yuj6@wit.edu

Specialty section:
This article was submitted to Human-Media Interaction, a section of the journal Frontiers in Computer Science

Received: *13 November 2020*
Accepted: *22 July 2021*
Published: *05 August 2021*

Citation:
Yu C-H, Shelton Z, Abou Nassif Mourad O and Oulal MA (2021) Readability Enhancement for PDF Documents. Front. Comput. Sci. 3:628832. doi: 10.3389/fcomp.2021.628832

INTRODUCTION

About five thousand years ago, the first mature writing systems were introduced by Egyptian and Sumerian scholars. Appropriate symbols were used to represent word's phonology (McGuinness, 2004). The writing systems keep evolving and the humans pass knowledge to next generation with it. To learn this knowledge, reading is an essential behavior. Although reading can be considered as an easy task for many people, it is not true for some people, such as people with dyslexia, people with visual impairment, the readers whose first language is not the same as the one used in the written article, etc.

In terms of improving reading in general, many topics have been explored in the past, such as recognition of cohesive ties in reading (Williams, 1983), interaction techniques [navigation in dual display (Chen et al., 1779), active reading support (Schilit et al., 1998), etc.] and factors influencing

reading [reading goal vs. reading time and comprehension (Protopsaltis and Bouki, 2006), the effectiveness of reading on screen (Gould et al., 1987), etc.]. Instead of traditional approaches, some researchers proposed a new direction to transform the presentation of the content to enhance readability, such as VSTF (Walker et al., 2005) and *Jenga* format (Yu and Miller, 2010; Yu, 2012). Although content transformation is interesting and helpful in online reading, such as Web pages, these works are not yet applicable to be used in a fixed-layout document, such as Portable Document Format (PDF) format, which was developed by Adobe in the 1990s and has been widely used to present a document in its original text formatting on the Web. In nowadays, people have been using PDF documents frequently to exchange firsthand information. For example, technical report in business field and research papers published in academia.

On the other hand, a platform of having reading activity also evolves, ranging from tradition paper, PC, laptops, tablets to smartphones. Since mobile devices are getting cheaper and people used them to absorb information daily, we are specifically interested in this media. According to the GSM Association, the expected number of subscribers with mobile Internet will increase by 1.7 billion from 2017 to 2025 (The Mobile Economy 2018, 2018). Although it is a downward facing trend, 52.3% of Internet content reported being in the English language (Historical yearly trends, 2020). There is an estimate that 1.5 billion people will be learning English (Ammon, 2015) and there is a great potential for people to learn and interact with English articles on mobile devices. To maintain the original design of English articles, more and more authors try to use a fixed-layout format, such as PDF document, but this format presents a readability issue for some readers, such as non-native English readers and readers with a special need.

The research question we want to address is: Can we enhance the readability of PDF documents for people with different needs, such as people with visual impairment, dyslexia, and non-native English readers? In this paper, we have three contributions:

- We investigated the issue of reading PDF documents on mobile devices and implemented a prototype system, *PDFroggy*, to make content transformation possible on the fixed-layout documents.
- Based on the experience learned from building above prototype system, we proposed a new scalable framework to generalize the process for different user-defined configurations.
- We adopted the proposed framework to create another new PDF reading environment, *PDFroggy++*, to enhance PDF document readability on mobile devices.

In the following of the paper, we will start with the related work in the field, including paper reading, techniques and tools for reading, Web page reading and reading with PDF documents. *PDFroggy*, our first trial of applying content transformation to the PDF document, will be introduced. In *A Scalable Framework for Readability Enhancement on PDF Documents*, we will explain the new proposed framework and introduce a new system, *PDFroggy++*, that adopts this new proposed framework. Discussion, Conclusions and Future Work will be presented at the end of this paper.

ARTICLE TYPES

A-Type Articles: Original Research.

RELATED WORK

Since the first mature writing systems were introduced by Egyptian and Sumerian scholars in about five thousand years ago, reading has become a common behavior for people to learn knowledge (McGuinness, 2004). The research of reading has also evolved from understanding this behavior to how to teach reading (McGuinness, 2004) and how to help and support people to conduct this process (Walker et al., 2005; Yu and Miller, 2010; Yu, 2012). In terms of helping and supporting reading in general, many topics have been explored in the past. For example, researchers found that the recognition of cohesive ties in reading (Williams, 1983) can help paper reading.

Moving from traditional paper to digital format content, some factors influence reading, including reading goal vs. reading time and comprehension (Protopsaltis and Bouki, 2006), the effectiveness of reading on screen (Gould et al., 1987), etc. To support reading on a digital platform, new interaction techniques were proposed to support the reading, such as navigation in dual display (Chen et al., 1779), active reading support (Schilit et al., 1998), etc.

Instead of traditional approaches, some researchers proposed a new direction to explore the reading domain in electronic or hypertext reading, i.e., transforming the presentation of the content to enhance readability for the user groups, such as visual-syntactic text formatting (VSTF) (Walker et al., 2005) and *Jenga* format (Yu and Miller, 2010; Yu, 2012). Previous work (Walker et al., 2005; Yu and Miller, 2010; Yu, 2012) have shown that content transformation can help enhance readability for specific users, such as children or non-native English readers. Both works conducted extensive user studies to verify readability from different directions, such as reading comprehension, reading speed, user satisfaction, etc. However, the studied content are online reading and Web pages, not fixed-layout documents. There was one unexplored domain in daily reading, i.e. PDF documents, which have been widely used for information exchange.

As defined in IETF RFC 3778 (- The application/pd), "PDF, the 'Portable Document Format', is a general document representation language that has been in use for document exchange on the Internet since 1993." Because it is popular, this format was standardized as an open format in 2008 (O 32000-1:2008 - Docume, 2000). It has a full set of features, including embedded fonts and images, a small file size and the ability to look the same on different platforms. PDF is a static format that elements stay at the same location no matter the screen or platform changes. Compared with hypertext contents,

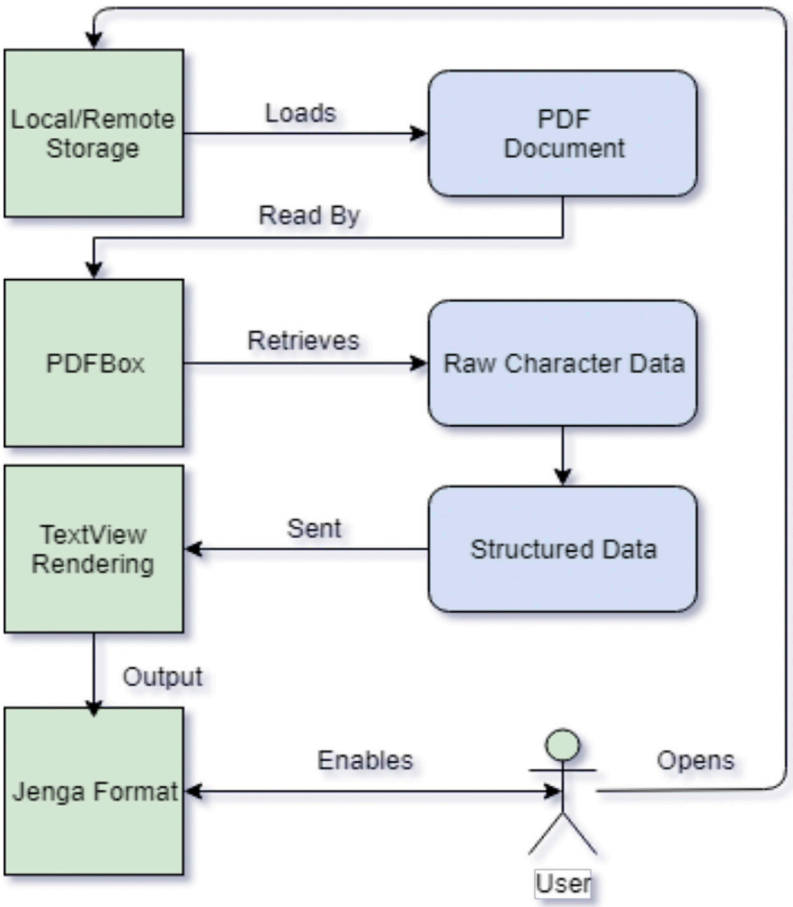


FIGURE 1 | The process of applying content transformation to a PDF document.

which are written in HTML and can be manipulated by using JavaScript with Document Object Model (DOM) APIs (N web docs - Document O), PDF documents do not have such interfaces to programmatically control the presentation. Although MaxTract (Josef and Volker, 2012) and Infty (Suzuki et al., 2003) shared a similar idea of re-engineering the transformation for PDF documents, they are mainly for Mathematics and accessibility, not readability enhancement per se.

In terms of usability and accessibility, Çakir's (Çakir, 2016) work identified issues with PDF documents and proposed solutions to address them, i.e. altering an article to enhance the readers' experience depending on their preferences. The proposed solutions include using Optical Character Recognition (OCR) to scan and transform a PDF document to .doc files and improving the search functionality on multi-language PDF documents. While these solutions embark on improving the usability and accessibility of PDF documents, they do not provide readability support for the readers, such as improving the readers' comprehension. Fundamentally, the contents of a PDF file are absolute, and each letter is arranged strictly (Adobe Systems Incorporate). This is perfect if the intention is to preserve layout. However, there is no guarantee the content is laid out in a way to maximize readability. Our idea of improving readability of PDF documents was based on previous work (Walker et al., 2005; Yu and Miller, 2010; Yu, 2012) that scientifically proved that content transformation can be used to enhance reading comprehension and extended to PDF documents.

PDFROGGY—CONTENT TRANSFORMATION FOR PDF DOCUMENTS

In this section, we would like to introduce our first prototype system, *PDFroggy*, which focused on enhancing readability for PDF documents on mobile devices (Shelton and Yu, 2020).

System Design

Based on the results done by (Walker et al., 2005; Yu and Miller, 2010; Yu, 2012), we believe content transformation can provide a way to enhance readability for PDF documents. We studied and investigated a way to apply content transformation to a PDF

1) Initialize a Document object, $D = \{Pi \mid i = 1 \ldots n\}$, where P stands for each page in the document.
2) For each page, *Pi:*
   a) *Add a new Page, Pi, to D*
   b) *Initialize objects Paragraph curPar, Sentence curSent and Word curWord*
   c) *Initialize int minX and minY, to the value of* $2^{31}-1$
   d) *Initialize int maxX and maxY to the value of* -2^{31}
   e) *For each character, C*
      i) *posX < minX | minX=posX, posX > maxX | maxX = posX*
      ii) *posY < minY | minY=posY, posY > maxY | maxY = posY*
      iii) *C is whitespace and C + 1 is not whitespace*
         (1) *Add C to curWord, curWord to curSentence*
         (2) *Set curWord to new Word object*
      iv) *C is ending punctuation*
         (1) *Add curWord to curSentence, set curWord to new Word object*
         (2) *Add curSent to curPar*
         (3) *Set curSent to new Sentence object*
      v) *C + 1 posY > C posY + C height and C + 1 posX > minX + C width*
         (1) *Add CurPar to curPage*
         (2) *Set curPar to new Paragraph object*
   f) *For each paragraph, Pa*
      i) *For each sentence in Pa, S*
         (1) *Initialize empty List object, tempWordList, tempList*
         (2) *Initialize new TextPosition object, lastPos*
         (3) *Copy contents of sub-sentence list in S over to tempList*
         (4) *Clear sub-sentence list in S*
         (5) *For each word in tempList, W*
            (a) *Add W to tempWordList*
            (b) *Is end of line*
               (i) *Initialize LineSplice object, Ls*
               (ii) *For each word in tempWordList, Wo*
                  1. *Convert Wo to a SimpleWord, add to Ls*
               (iii) *Add Ls to sub-sentence list in S*
               (iv) *Clear tempWordList*
            (c) *Set lastPos to last character in W*

FIGURE 2 | The algorithm of extracting document structure.

document, which does not have DOM APIs-like design to manipulate content presentation. In addition, we are interested in using mobile device as a platform.

The main design idea is to load and decode the PDF document and then apply a specified transformation to the rendered content. *PDFroggy* is an Android application that demonstrates the idea. *PDFroggy* allows the user to open up and display a PDF document on their mobile device. Then, it can apply a specified transformation method to the PDF content, either in a paragraph or sentence level, to re-render the content (**Figure 1**).

System Implementation

Parsing the PDF Document

There are different Java libraries available to be used to parse and render PDF files. Our first attempt was to use default APIs built in Android SDK. As part of API level 21, Android provided a native solution for displaying PDFs, named PdfRenderer (AndroidK - PdfRenderer). However, it does not allow for any editing functionality to the rendered content. This limitation makes content transformation infeasible.

Instead, we chose Apache's PDFBox, which is a library that allows for getting detailed information of a PDF document. Although PDFBox uses classes from the java.awt package, these classes are not included in the distributed Android SDK libraries. Fortunately, PdfBox-Android, an Apache PDFBox project ported to work on Android, replaces java.awt with compatible Android libraries such that PDF documents can be read and parsed by PDFBox (PdfBox-Android) on Android devices.

Basically, PDFBox provides a functionality for text stripping, but the challenge of making content transformation available on a PDF document is to recreate a new PDF document with a desired format. This recreation process needs more information, including font family, weight, size, character location, etc. This information isn't readily available to the programmer, as it remains protected within PDFBox's TextStripper class. To access it, we created a child class of TextStripper class and

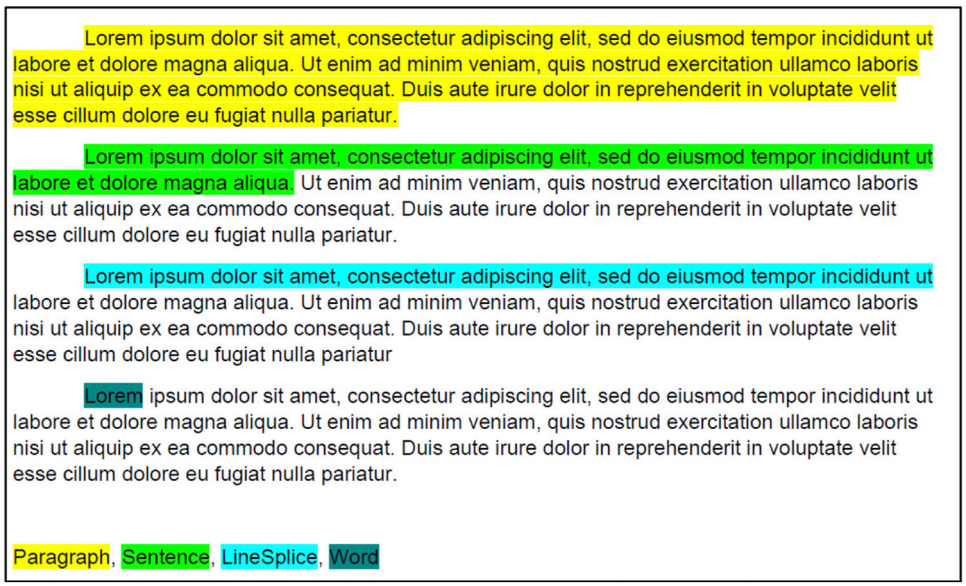


FIGURE 3 | *PDFroggy*—Data structure visual breakdown.

placed all required information for having a new PDF document into this child class.

Extracting Document Structure

After parsing the PDF file, the next step is to extract the layout information from the file such that we can separate sentences and apply content transformation. Unfortunately, because PDFBox does not provide sentence or paragraph-level information, we need to reconstruct this information from character-level information. The algorithm designed and used is illustrated as **Figure 2**.

When the system finishes this process, i.e. the first iteration of the characters, the data will be split into a more usable data structure like: Document → Page → Paragraph → Sentence → Word [Word/(LineSplice → SimpleWord)]. It achieves this by using simple needle and haystack techniques to look for punctuation and change in character location. The data structure also maintains some import meta-data that is, extracted from PDFBox, such as font size, font family and weight. In addition, it also stores the exact position information for each character such that the document can be rendered on the mobile device with an accuracy (**Figure 3**).

During the first iteration, the system also stores the maximum and minimum x and y position. Based on this information, the margins of the document are known for the second iteration. During the second iteration, it uses the previous data structure and other information to convert it to: Document → Page → Paragraph → LineSplice (SubSentence) → SimpleWord format.

Rendering Content With Native APIs

To render the text stripped from the PDF document, we tried different UI components and decided to use the simplest one, i.e. TextView. TextView is a native text rendering component in Android and is created for each LineSplice.

Each SimpleWord is added to a SpannableStringBuilder that can provide different styles to each individual word within a LineSplice. Each of these generated TextViews gets its location based on its original location in the PDF file. To provide flexible screen size, these TextViews are added into a ScrollView element. The TextView is assigned an index-based ID for each structure level. The format is similar to the following:

(page_no): (paragraph_no): (sentence_no): (splice_no)

Now that each TextView has an assignable address within the rendered screen, the content transformation method can take over and modify the contents of the PDF displayed on the screen.

Applying Content Transformation

There are different types of content transformation methods and they can be applied to the rendered content, such as naïve sentence separation (making sentences separated and presented at different lines), VSTF (Walker et al., 2005), *Jenga* format (Yu and Miller, 2010), etc. To demonstrate the effect, *PDFroggy* uses *Jenga* format as an example (**Figure 4**). In the current design and implementation, the content transformation method is abstracted out to a separate file so that new and different content transformation methods can be added. This mechanism makes the system scalable for the future extension.

A SCALABLE FRAMEWORK FOR READABILITY ENHANCEMENT ON PDF DOCUMENTS

Lessons From Previous Work

Although our first prototype system, *PDFroggy* (Shelton and Yu, 2020) (*PDFroggy–Content Transformation for PDF documents*), was able to address the readability issue by implementing a

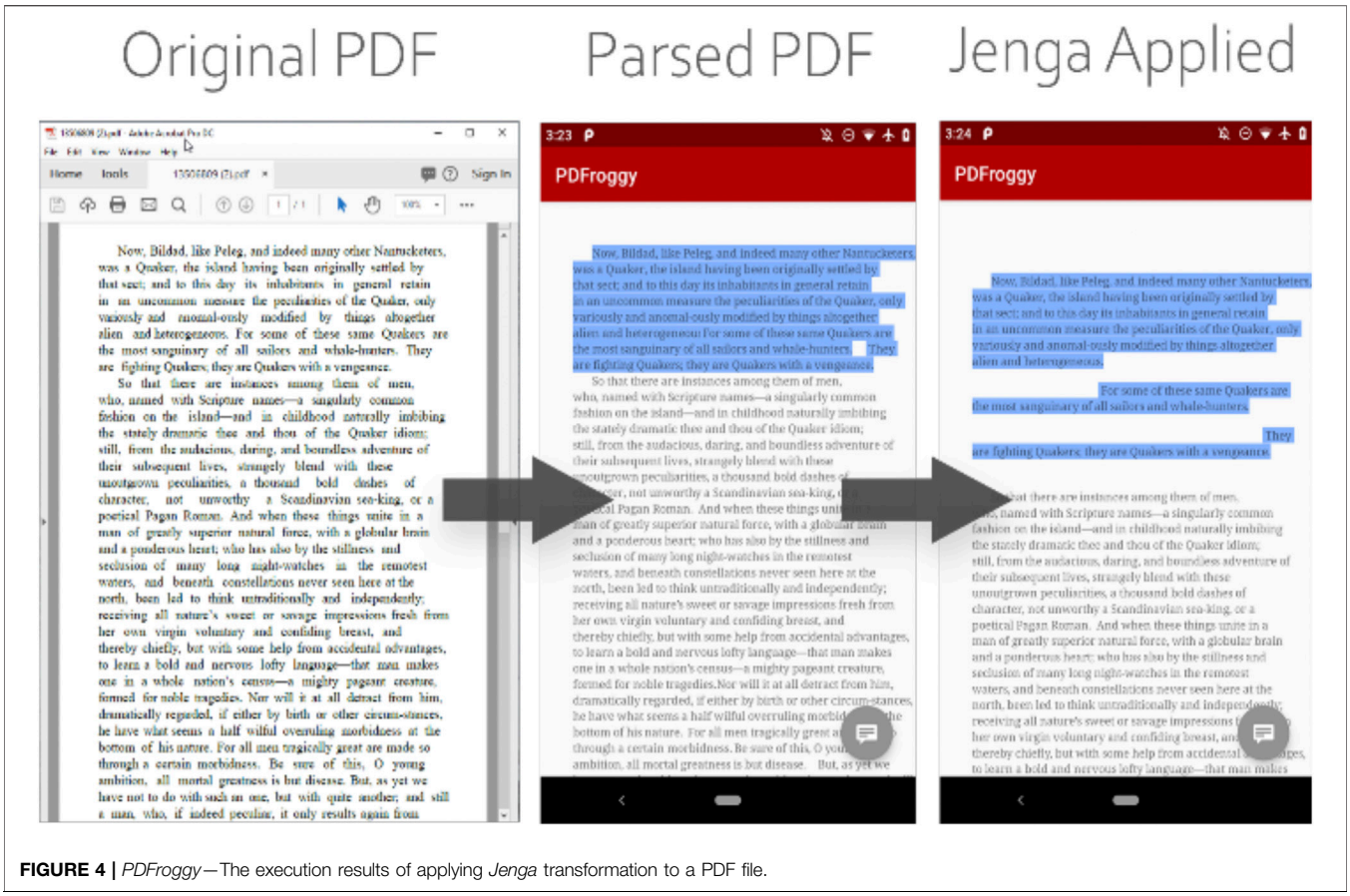


FIGURE 4 | *PDFroggy*—The execution results of applying *Jenga* transformation to a PDF file.

transformation method, this results in a considerable sacrifice in accessibility.

The lessons learned from our previous project are three folds. Firstly, compared with regular web pages, which are normally written in HTML and can be manipulated by JavaScript with DOM APIs, PDF documents were not designed with such interfaces to programmatically control the presentation. Secondly, even if we can have such interfaces to control the presentation, the users' preferences cannot be dynamically configured for PDF documents. In addition to VSTF format (Walker et al., 2005) and *Jenga* format (Yu and Miller, 2010), we believe there will be more useful and creative content transformations for people with special needs. Thirdly, in terms of comfortability level in reading, *PDFroggy* was not able to address it. For example, some people like to read content in a contrast environment (white text and black background), but some people prefer a configured environment, such as a larger font size, double spacing, or keywords highlighted.

Based on above lessons, we believe there is a need to design a scalable framework for readability enhancement on PDF documents. The goal of the framework is not only to generalize the users' needs into multiple levels, but it should also be applicable to PDF documents and configurable for mobile platforms, such as smartphones and tablets.

Framework

Previous work has demonstrated that content transformation (Walker et al., 2005; Yu and Miller, 2010; Yu, 2012) is a way to enhance readability. We also believe the users' needs or preferences are another important factor, such as font size, font family, contrast, keyword highlighting . . ., etc. Therefore, in addition to content transformation, we take user customization into the framework design. It is important that the application built with the framework can accommodate any combination of the users' preferences and content transformation. In terms of the users' preferences, the users should be able to customize their preferences with three levels of content formatting, including paragraph level, sentence level, and word level (such as each sentence's leading word). Each level has a potential to be customized further as one would normally do in rich text. In addition, we also want to introduce the concept of dynamic transformations. These are special formatting options, defined by the user, that only occurs upon tapping a paragraph. All existing transformation methods, such as VSTF and Jenga format, can be part of this design. Aside from one more layer of customization, this introduces a layer of interaction to the viewing content. The framework we proposed allows the users to develop their own formatting preferences that are stored in a JavaScript Object Notation (JSON) (Introducing) format file. JSON can be considered as a light-weight data-exchange

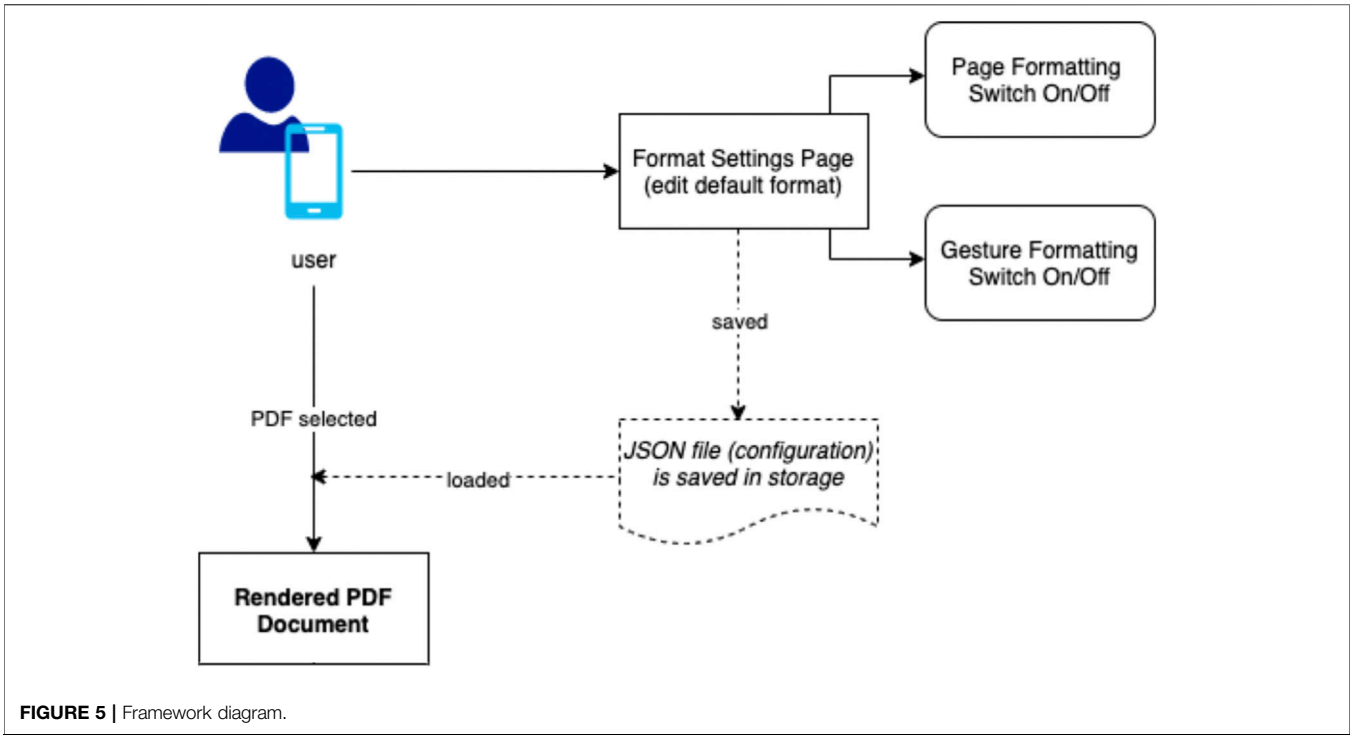


FIGURE 5 | Framework diagram.

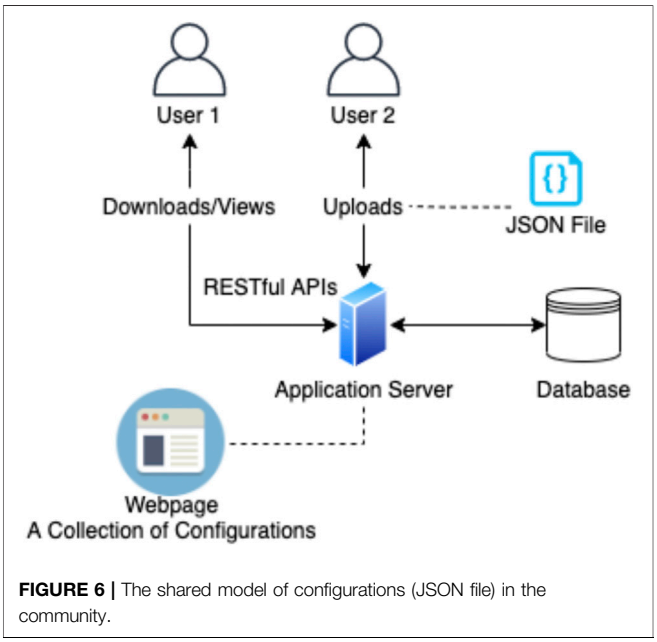


FIGURE 6 | The shared model of configurations (JSON file) in the community.

format and it has been used to encode information in structured text. Rather than writing their own JSON file, the users can create preferences on a user-friendly setting page to indicate how they want the content to be rendered or formatted. For example, the user can decide the font color, background color or font family of the PDF document. The users can also define two different formatting preferences: page formatting and gesture formatting. While the page formatting changes the layout structure of the PDF document, such as headers, paragraphs and sentences, the gesture formatting is to change the format of the selected area that the users interact with. This can be done with gestures such as tapping a paragraph. Both page formatting and gesture formatting are able to be toggled respectively. For example, if the users decide to view the original document but still want to format particular content with gesture formatting, they are able to toggle the page formatting off and keep the gesture formatting on. The framework diagram is illustrated in **Figure 5**.

We also believe a good configuration (JSON file) should be shared to benefit people with the same need, such as people with visual impairment, dyslexia and non-native English readers. The framework considering the model of sharing configurations is illustrated in **Figure 6**.

PDFroggy++—An Implementation for Proposed Framework

Based on the proposed framework, we are interested in knowing if it is possible to implement a new PDF reading environment to demonstrate some of the proposed ideas. In this section, we would like to introduce the details of the design and implementation of such new prototype system, *PDFroggy*++, which is an Android application with SDK version 11 (API level 30).

Content Loading—The PDFContentManager

Inspired by the *PDFroggy* (*PDFroggy–Content Transformation for PDF documents*), *PDFroggy*++ continues to use Apache's PDFBox (PdfBox-Android) for parsing PDF documents. However, we found the document processing was time

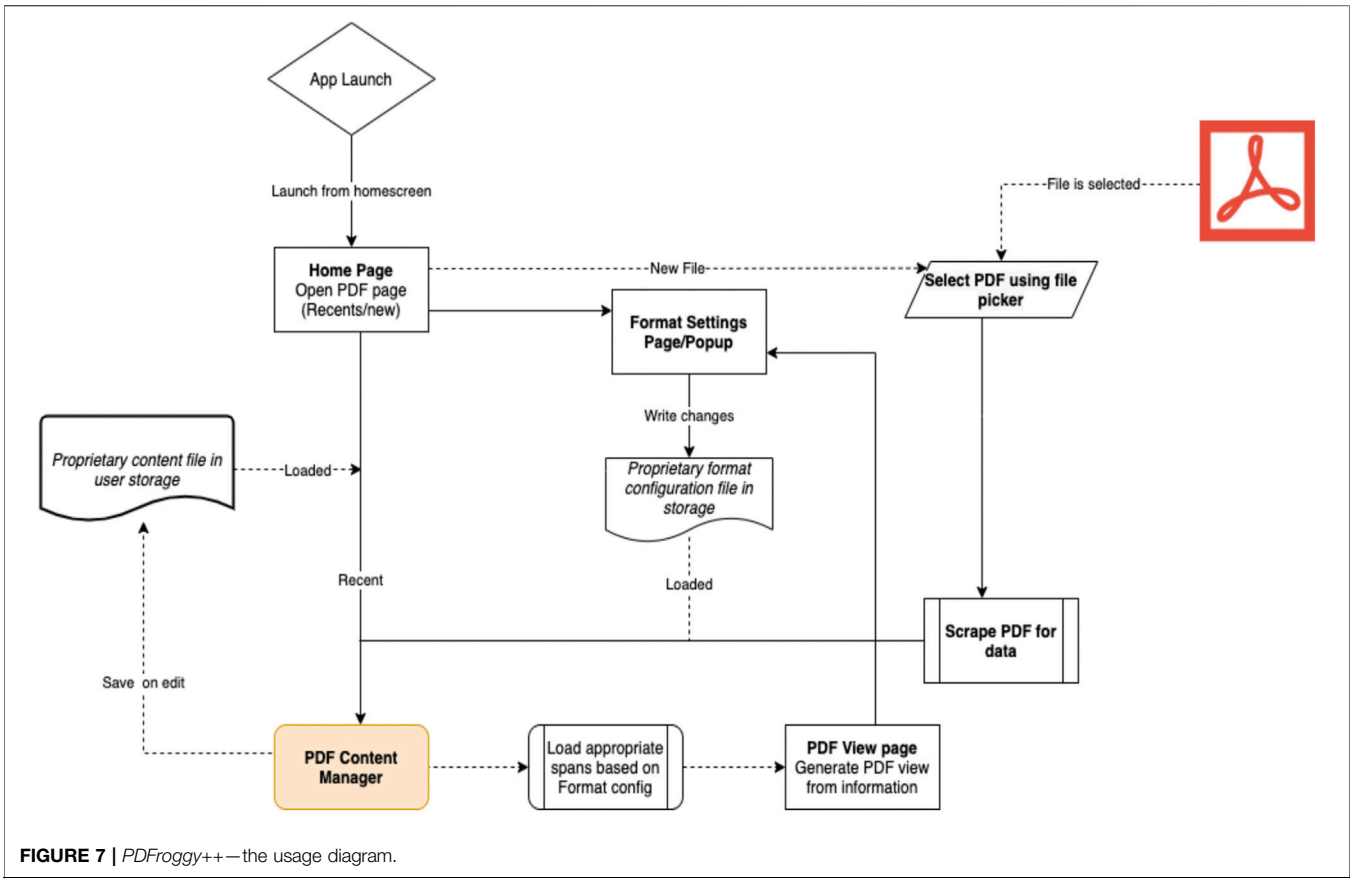


FIGURE 7 | *PDFroggy++*—the usage diagram.

consuming in *PDFroggy* project. In *PDFroggy++*, we addressed the issue by creating the PDFContentManager, a singleton running on a separated thread, and the performance of this approach dramatically improved the processing time. After the investigation, we found the root cause of the lagging mainly comes from the interruptions of PDDocument.load () to the main UI thread. The PDFContentManager is responsible for all operations related to the preprocessing of PDF document.

Text Striping With InfoScraper

As for the actual extraction process, it is handled entirely by a custom InfoScraper object which extends PDFBox's PDFTextStripper and overrides the writeString () method. Similar to the method found in the default PDFTextStripper, InfoScraper's writeString () builds a string encompassing all text found in a page. However, whereas default behavior of the function would return all text separated by line, the overridden method returns the text separated by paragraph, which is important for readability enhancement since readers normally read articles paragraph by paragraph and is easier for interaction design.

Text Formatting With FormatterObjects

Upon the completion of the loading and text extraction processes, the PDFContentManager records the boundaries of each paragraph, sentence, and leading word and then applies transformations to these spans in accordance with the user's preferences. This process is done via the instantiation of FormatterObjects, objects tasked with applying ParceableSpan objects to the text in accordance to the user's preferences. The PDFContentManager instantiates a FormatterObject object for every span and keeps a reference to all of them such that it can undo the transformation when necessary.

Content Layout and Display

In *PDFroggy*, we treated the viewer as a document and essentially recreated the PDF document using Android TextView objects arranged, character by character, in such a way that they respect the original layout of the document. This is a perfectly sound approach as the result of this is a similar PDF view in which each of the original text objects, each contained in separated TextViews, can be transformed dynamically.

However, *PDFroggy++* does not recreate PDF documents but instead uses the same layout information its predecessor used to arrange the TextView objects to construct a single string in which text from the appropriate text objects in the PDF document can share paragraphs. This text is then represented by a single TextView object built from a SpannableString, effectively circumventing the need to follow the original PDF's layout information.

This change brings in an advantage of offering a flexibility. Since the original absolute positions play no role in

A

```
"page_format": {
   "paragraph": {
      "separate_by": true,
      "spacing": 2,
      "margin": 2,
      "font_size": 14,
      "background_color": "GRAY"
   },
   "sentence": {
      "spacing": 2,
      "margin": 2
   },
   "word": {
      "spacing": 0
   },
   "letter": {
      "spacing": 0.2,
      "font_color": "WHITE",
      "font_style": "default"
   },
   "image": {
      "show_images": true,
      "spacing": 1,
      "margin": 2
   },
   "accessibility": {
      "read_page_option": false
   }
   "grammar": {
      "highlight_key_words": false,
      "highlight_key_sentences": false
   }
},
```

B

```
"gesture_format": {
   "separate_by_sentence": true,
   "letter_spacing": 0,
   "word_spacing": 0,
   "sentence_spacing": 1,
   "line_spacing": 1,
   "load_predefined_format": "Jenga"
}
```

FIGURE 8 | An example of JSON file used for configuring content presentation **(A)**: Page Formatting, **(B)**: Gesture Formatting.

determining where TextView objects are rendered in *PDFroggy++*, the need to respect margins as text would be in a document is not there. This means that *PDFroggy++* is inherently more capable of dynamically utilizing a wider range of formatting options since it does not have to follow the rules of a document when transformations are applied, such as using a smartphone or tablet, displaying content in portrait mode or landscape mode, etc. Additionally, the existence of spannable text objects allows a high-level dynamic formatting.

Shared JSON for Community

There are two formatting methods defined: page formatting and gesture formatting. For the gesture formatting, a user can also load a JSON file that has more specific configurations for the interaction on the viewing page. An example of these configurations is the *Jenga* format (Walker et al., 2005; Yu and Miller, 2010). The users can create a specific JSON file that defines how they want to format the text to make it more readable. In *Jenga* format, the user can change the sentence structure or environment to enhance readability, such as increasing the spacing between sentences or making a strong contrast for the specified area. We plan to create a community to allow the users to share defined configuration (JSON files) via using RESTful APIs in the server and HttpURLConnection in the Android side (**Figure 6**). The format files can be created by the users or the format developers.

RESULTS

The usage diagram of *PDFroggy++* is illustrated in **Figure 7**. It starts from application launch to load PDF documents and configurations, and then render the document. The screenshots of current implementation of *PDFroggy++* can be found in **Figures 8–11**. As **Figure 8** illustrates, the left-hand side is an example of page formatting and the right-hand side is an example of gesture formatting. **Figures 9**, **10** show the presentation changes of a PDF document after applying page formatting and gesture formatting respectively. **Figure 11** is the settings page for content presentation. In short, *PDFroggy++* follows the proposed framework to implement a new reading environment for PDF documents, which can be customized by using different configurations (JSON files) to have page formatting and gesture formatting. While page formatting is used to customize page level customization, gesture formatting can be used to customize interaction on selected area. For example, the content transformation methods proposed by (Walker et al., 2005; Yu and Miller, 2010; Yu, 2012) can be encoded into gesture formatting to provide interaction on the viewing page.

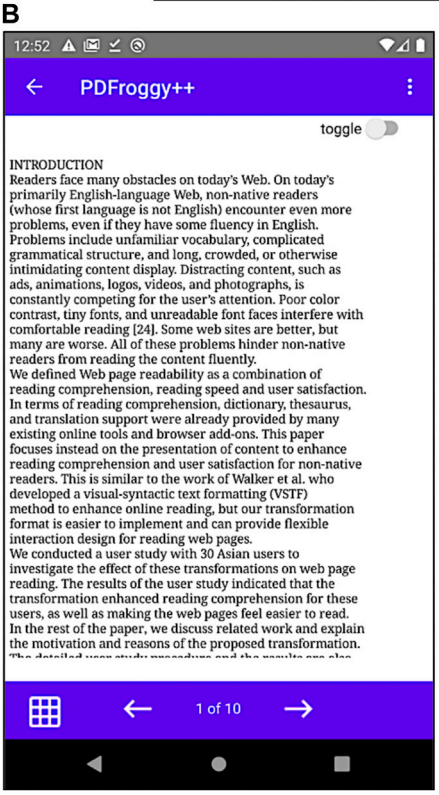
DISCUSSION

Applying content transformation to enhance readability has been studied in the past (Walker et al., 2005; Yu and Miller, 2010; Yu,

A

```
"page_format": {
   "paragraph": {
      "separate_by": true,
      "spacing": 2,
      "margin": 2,
      "font_size": 14,
      "background_color": "GRAY"
   },
   "sentence": {
      "spacing": 2,
      "margin": 2
   },
   "word": {
      "spacing": 0
   },
   "letter": {
      "spacing": 0.2,
      "font_color": "WHITE",
      "font_style": "default"
   },
   "image": {
      "show_image": true,
      "spacing": 1,
      "margin": 2
   },
   "accessibility: {
      "read_page_option": false
   },
   "grammar": {
      "highlight_key_words": false,
      "highlight_key_sentences": false
   }
}
```

B



C

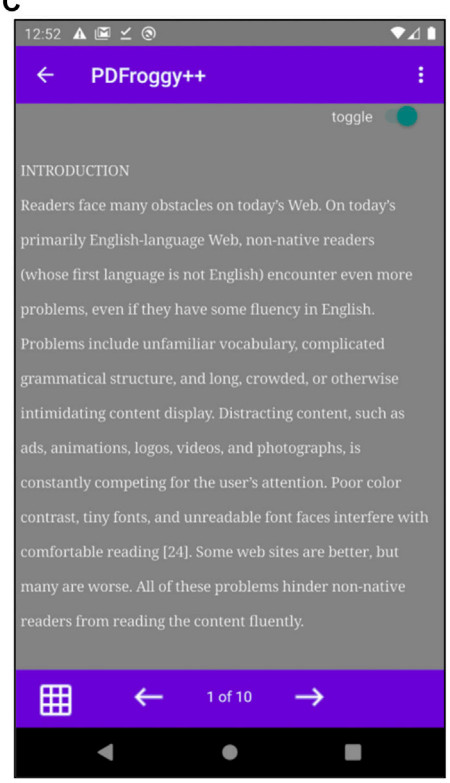


FIGURE 9 | (A) An example of page formatting JSON file; **(B)** Original PDF view; **(C)** Formatted content.

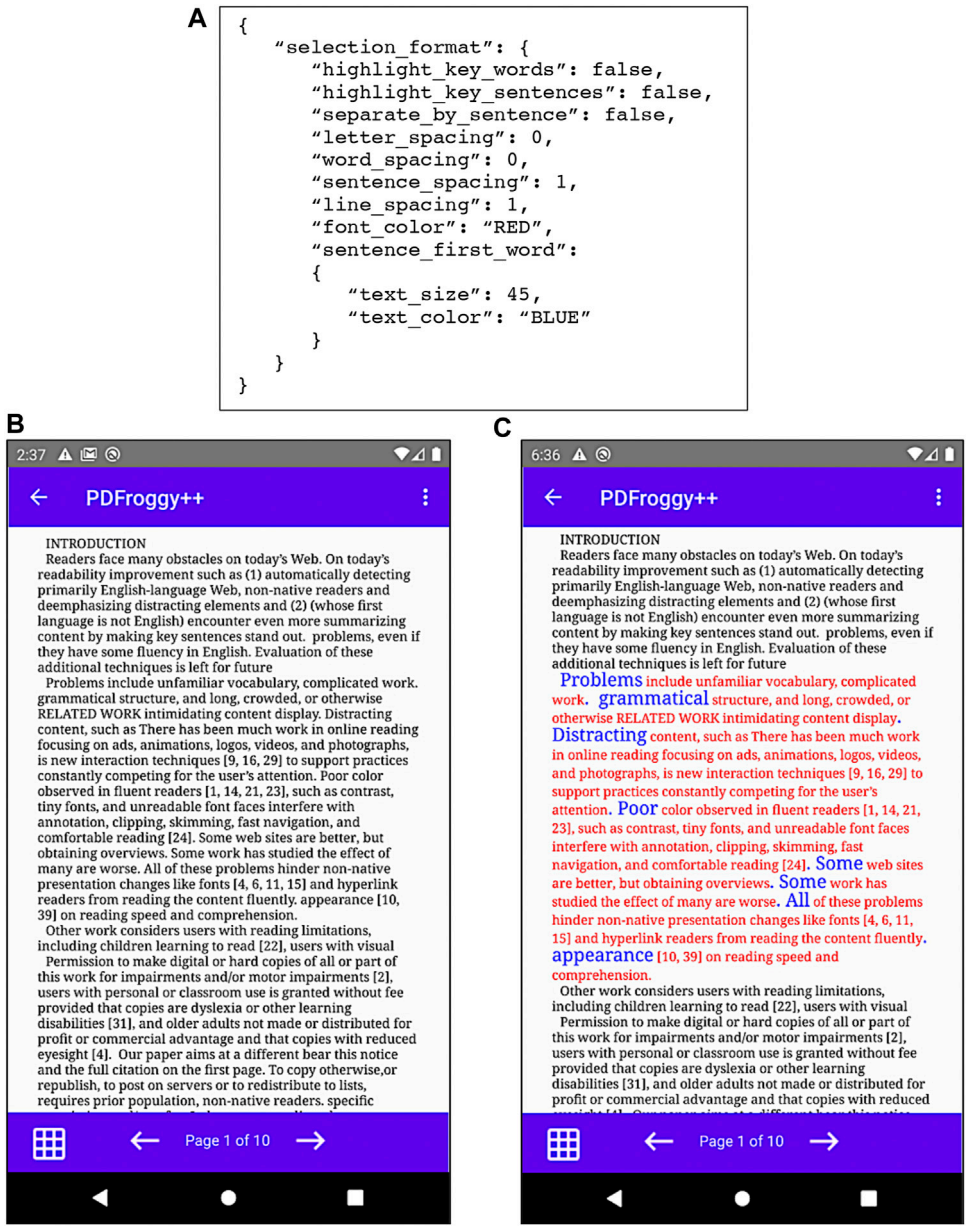


FIGURE 10 | (A) An example of gesture formatting JSON file; **(B)** Original PDF view; **(C)** Formatted content of selected paragraph—leading characters are enlarged and marked with blue colors, and selected paragraph is marked with red color.

2012), but the idea was not yet extended to a PDF document, which presents more restrictions in its format. Generally speaking, PDF document is read in the Adobe Acrobat Reader or within the browser. Although we can transform a PDF document to HTML document and apply transformation methods in the end, we believe creating a suite in which the user can dynamically transform the content of a PDF document on mobile devices can provide a better accessibility for the mobile users.

PDFroggy demonstrates that we are able to apply content transformation to a PDF document, but it is still a standalone mobile application and not yet integrated into a mobile Web browser. Part of the reasons is that there is no browser extension mechanism available in default mobile Web browser. Even existed mobile Web browsers, such as Chrome, Opera, Firebox, etc., not all of them provide extensions mechanism to customize the web pages, not mention fixed-layout PDF documents. However, it is possible to extend mobile applications to support Web accessibility by adding WebView component, which is a browser engine that can be used to render web contents and merge the results into one presentation. *PDFroggy* is our first attempt to address PDF readability from

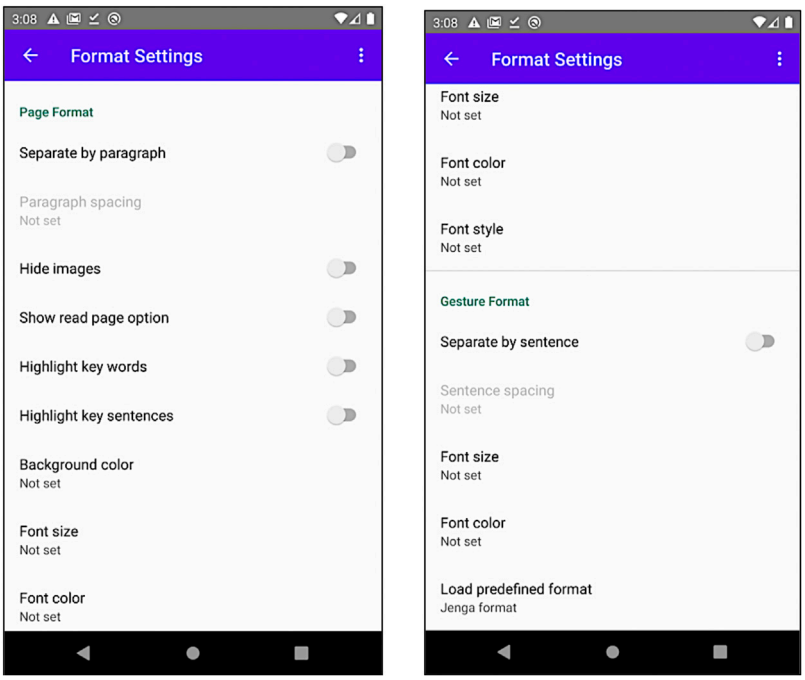


FIGURE 11 | Settings for content presentation.

application perspective. There is need to improve current rendering such that it can be integrated into regular mobile Web content.

Based on the learning from *PDFroggy*, we designed a new framework that considers content transformation *via* page formatting or gesture formatting. *PDFroggy++* uses the proposed framework and expands the idea of formatting PDF documents to make them readable. Furthermore, it introduces customization options to reformatting PDF documents. These customization options help the users reformat documents for the need and improve the readability and accessibility of PDF documents.

One special topic we not yet able to address is multiple columns and embedded multimedia. As we notice, academic research papers or journal articles are written in multiple columns and some news or textbook-like PDF documents might contain multimedia data, such as audio, video, and interactive content. In terms of multiple columns, unfortunately, PDFBox does not provide enough information to reconstruct the document. All of them need further investigations.

CONCLUSIONS AND FUTURE WORK

PDF document format was designed to maintain its fix-layout content for delivering information. Although it can keep the original structure, this format presents usability, and accessibility issues (Çakir, 2016). Many techniques and methods were created in the past for paper and online (including Web pages) readings, but none of them can be used directly for PDF documents. This research aims at enhancing readability for PDF documents for people with different needs.

Parsing a PDF file is challenging but recreating a new PDF document with a desired content transformation for readability enhancement is meaningful for some people, especially there are more and more people using PDF format to deliver first-hand information written in English. In addition, different content transformation methods are beneficial to different user groups, such as VSTF format for children, *Jenga* format for non-native English reading, etc. We believe more and more useful transformation methods will be proposed. Therefore, our first attempt, *PDFroggy* (Shelton and Yu, 2020), was to shorten a gap between static PDF document and customizable PDF content on mobile devices for different user groups. To the best of our knowledge, this was the first research that targets at the readability of PDF documents on mobile devices, and more specifically it tried to apply transformation method to the content to enhance the readability and accessibility for readers with a special need, such as ESL readers.

However, there are some limitations in this work (Shelton and Yu, 2020). For example, the current results are good for text-heavy articles lacking complex document structures, such as multiple columns, embedded multimedia, tables, etc. Supportive and meta information (footnotes, headers, footers, etc.) are not yet covered. If a PDF document uses embedded fonts, which aren't shipped as parts of the mobile operating system, the content rendering can only use default fonts. The content transformation method is not flexible to extend to support other usability and accessibility needs. In terms of the

performance, there was an unexplainable lag spike caused by PDFBox in *PDFroggy*.

We learned from our work (Shelton and Yu, 2020) and started to redesign a scalable framework, which not only allows the users to configure their transformation need, but it also provides different types of transformation, i.e. page formatting vs. gesture formatting. Based on this framework, we were able to re-implement another system, *PDFroggy++*, to demonstrate the ideas. In *PDFroggy++*, the users can define their preferences as a configuration into a JSON file (Introducing) and then load it to provide content transformation to the viewing page. In addition, the design of *PDFroggy++* allows the users to share the configurations in the community. The users will be able to upload their JSON formatting files, view and download the interested files. After downloading the desired JSON file, the users can load the file *via* the settings page to see the content transformation.

The current result indicates that it is possible to apply content transformation to enhance PDF readability on mobile devices. We believe it is worth further investigations to make PDF documents readable and accessible for different people with different needs, such as non-native English readers, people with dyslexia or special needs, etc. Although we still have a long way to achieve the ultimate goal, the current result is a good foundation to explore related issues further.

In the future, there are two directions of this research. Firstly, we will continue to improve *PDFroggy++* and refine the framework. As mentioned in Discussions, one special topic we not yet able to address is multiple columns and embedded multimedia. Although current implementation demonstrates the possibility of applying content transformation to a simple PDF document, some materials, such as academic research papers or journal articles, are written in multiple columns and some news or textbook-like PDF documents might contain multimedia data, such as audio, video, and interactive content. All of these topics need further investigations to make *PDFroggy++* useful. Secondly, we would like to evaluate the acceptability of the framework for people with different needs. To achieve this goal, we need to understand how certain modifications to PDF documents can help specific accessibility issues, such as people with dyslexia, visual impairment, and non-native English readers. As long as we can understand this topic better, we can integrate these modifications into the configurations to make the system easy to use.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

AUTHOR CONTRIBUTIONS

C-HY was the principal investigator of this research and responsible for ideas of the research, framework design, systems design and implementation support. ZS contributed to design and implement the first prototype system, *PDFroggy*, to apply a content transformation to PDF documents. OANM and MO co-designed a scalable framework with C-HY and implemented a new prototype system, *PDFroggy++*, for the Android platform. All authors contributed to the article and approved the submitted version.

ACKNOWLEDGMENTS

The authors would like to thank Wentworth Institute of Technology to provide an opportunity for all authors to know each other and work together as a team to address an identified research topic.

REFERENCES

Adobe Systems Incorporated, "Adobe Portable Document Format Version 1.4", PDF Reference, third edition, pp. 9–12.

Ammon, U. (2015). The Status of the German Language in the World, Cited in Noack, R and Gamio, L (2015) the World's Languages, in 7 Maps and Charts Washington Post 23 April 2015. Available at: https://www.washingtonpost.com/ news/worldviews/wp/2015/04/23/the-worlds-languages-in7-maps-and-charts/?utm_term=.c7342219767b.

Android SDK - Pdf Renderer Class. Available at: https://developer.android.com/reference/android/graphics/pdf/PdfRenderer.

Çakir, A. (2016). Usability and Accessibility of Portable Document Format, *Behaviour & Information Technology*, 35 Issue 4. , p324–334. doi:10.1080/0144929x.2016.1159049

Chen, N., Guimbretiere, F., Dixon, M., Lewis, C., and Agrawala, M. (1779). Navigation Techniques for Dual-Display E-Book Readers. Proceeding of the Twenty-Sixth Annual CHI Conference on Human Factors in Computing Systems (CHI '08). New York, New York, USA: ACM Press.

Gould, J. D., Alfaro, L., Finn, R., Haupt, B., and Minuto, A. (1987). Why reading Was Slower from CRT Displays Than from PaperSIGCHI Bull. In CHI '87 Proceedings of the SIGCHI/GI Conference on Human Factors in Computing Systems and Graphics Interface, 18. New York: ACM Press, 7–11. doi:10.1145/1165387.30853

Historical Yearly Trends in the Usage Statistics of Content Languages for Websites, W3Techs (Web Technology Surveys), 2020. Available at: https://w3techs.com/technologies/history_overview/content_language/ms/y

IETF - the Application/pdf Media Type. Availableat: https://tools.ietf.org/html/rfc3778

Introducing, J. S. O. N. Available at: https://www.json.org/json-en.html.

ISO 32000-1:2008 - Document management - Portable Document Format - Part 1: PDF 1.7. Available at: https://www.iso.org/standard/51502.html

Josef, B., Alan, P. S., and Volker, S. (2012). MaxTract: Converting PDF to LATEX, MathML and Text." *AISC/DML/MKM/Calculemus. 7362 of Lecture Notes in Computer Science*, volume. Springer, 422–426.

McGuinness, D. (2004). *Early Reading Instruction - what Science Really Tells Us about How to Teach Reading*. Cambridge, Massachusetts: The MIT Press. doi:10.7551/mitpress/2545.001.0001

MDN Web Docs - Document Object Model (DOM): Available at: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

The Mobile Economy 2018, 2018 GSM Association. Available at: https://www.gsmaintelligence.com/research/?file=061ad2d2417d6ed1ab002da0dbc9ce22&download

PdfBox-Android. Available at: https://github.com/TomRoush/PdfBox-Android

Protopsaltis, A., and Bouki, V. (20062006). The Effects of reading Goals in Hypertext reading. Proceedings of The24th Annual Conference on Design of Communication (SIGDOC '06), 29–34. New York, New York, USA: ACM Press. doi:10.1145/1166324.1166332, pages

Schilit, B. N., Golovchinsky, G., and Price, M. N. (1998). Beyond Paper: Supporting Active reading with Free Form Digital Ink Annotations. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '98). New York, New York, USA, January: ACM Press, 249–256.

Shelton, Z., and Yu, C-H. (20202020). A Framework for PDF Readability Enhancement. The 17th International Web for All Conference. W4A. doi:10.1145/3371300.3383352

Suzuki, M., Fumikazu, T., Fukuda, R., Uchida, S., and Toshihiro, K. (2003). Infty: an Integrated Ocr System for Mathematical Documents. In DocEng '03: Proceedings of the 2003 ACM Symposium on Document Engineering. New York, NY, USA: ACM Press, 95–104.

Walker, S., Schloss, P., Fletcher, C. R., Vogel, C. A., and Walker, R. C. (2005). Visual-Syntactic Text Formatting: A New Method to Enhance Online Reading. *Reading Online Electron. J.* 8 (6).

Williams, R. (1983). Teaching the Recognition of Cohesive Ties Tin Reding a Foreign Language. *Reading a Foreign Lang.* 1 (1), 35–53.

Yu, C-H., and Miller, R. C. (2010). Enhancing Web page Readability for Non-native Readers. "Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10). Atlanta, Georgia, USA. April 10-15, 2010. doi:10.1145/1753326.1753709

Yu, C-H. (2012). *Web Page Enhancement on Desktop and Mobile Browsers. Ph.D. Thesis*. Department of Electrical Engineering and Computer Science, MIT.

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.