

A Multi-Format Document-to-Speech Reader on Raspberry Pi for Accessible Digital Learning

Shaharil Mad Saad and Leh Wen Hao

Faculty of Mechanical Engineering, Universiti Teknologi Malaysia 81310 UTM Johor Bahru Johor, Malaysia

DOI: <https://doi.org/10.47772/IJRISS.2026.100800860>

Received: 4 September 2026; Accepted: 9 September 2026; Published: 19 September 2026

ABSTRACT

Blind learners frequently receive educational materials in heterogeneous digital formats, yet accessing those files can require a conventional computer, a screen reader, and multiple navigation steps. This paper presents a Raspberry Pi-based document-to-speech reader that directly processes TXT, PDF, DOCX, and PPTX files from removable storage through a compact Braille-oriented key interface. The Python implementation separates file navigation, format-specific text extraction, segmentation, keyword filtering, presentation control, file transfer, and speech synthesis into functional modules. Candidate extraction libraries were assessed with samples containing formatted prose, hyperlinks, Greek characters, equations, images, a resume layout, and presentation slides.

The comparison showed that docx2python retained the broadest range of DOCX content, including hyperlink and media markers, although equation conversion was imperfect. PyPDF2 produced the most faithful mathematical-text output among the tested PDF libraries, while python-pptx extracted the evaluated slide text without observed errors. Google Text-to-Speech was selected because playback could be interrupted within the implemented interaction flow. Functional verification confirmed file selection, sequential reading, keyword search, slide navigation, and file relocation using the assigned keys. The prototype demonstrates a practical architecture for portable access to common learning documents, while internet dependence, legacy DOC/PPT incompatibility, mathematical-symbol pronunciation, and conversion delay remain priorities for further development.

Keywords: assistive technology, blind learners, document extraction, Raspberry Pi, refreshable Braille, text-to-speech

INTRODUCTION

Digital course notes, past examination papers, textbooks, and presentation slides are commonly distributed as electronic files. This practice improves portability for many students, but it can create an additional access layer for blind learners when document formats, navigation controls, and assistive applications are not integrated. Refreshable Braille displays and screen readers can expose on-screen information, yet many commercial workflows still depend on a host computer and software-specific commands [1]–[3]. The practical problem addressed in this work is therefore not text-to-speech alone, but direct and consistent access to several common educational file formats through a small set of tactile controls.

Existing screen readers such as NVDA, JAWS, Dolphin Screen Reader, and VoiceOver provide broad operating-system access [5]–[8]. BRLTTY similarly supports Braille terminals in Linux environments [9]. These systems are powerful, but a learner may still need to locate a document, launch a compatible application, move through its interface, and manage different commands for different file types. Prior portable text-to-speech studies have focused primarily on camera or optical-character-recognition input [10]–[12], whereas the present project starts with documents that are already digital and stored on a USB drive.

The contribution of this paper is a system-level account and evidence-based component selection for a portable multi-format reader. The prototype integrates (i) direct extraction from TXT, PDF, DOCX, and PPTX files; (ii) Braille-oriented navigation and text entry; (iii) spoken output; (iv) keyword-based paragraph filtering; (v) PowerPoint slide control; and (vi) file relocation. Unlike the shorter preliminary manuscript, the evaluation reported here distinguishes functional verification from library comparison and documents the limitations observed in complex document content.

RELATED WORK AND DESIGN REQUIREMENTS

Accessible document interaction

A refreshable Braille display represents characters using mechanically actuated Braille cells and is commonly paired with a screen reader [1], [3]. Commercial devices differ in their ability to open files directly; some support BRL or TXT files from local storage, while richer formats are normally accessed through a connected host and screen-reader software. Kiyota et al. demonstrated a touch-based digital reading device for visually impaired users [4], illustrating the value of interaction models designed around accessibility rather than conventional visual interfaces. The target system was consequently defined by four requirements derived from the thesis: it should operate on a portable Raspberry Pi; accept a keyboard or Perkins-style Braille input; read the four prevalent formats used in the project (TXT, PDF, DOCX, and PPTX); and provide audio output without requiring a separate laptop. Additional functions—keyword search, slide navigation, and file relocation—were retained because they allow the user to locate and act on information rather than only listen from the beginning of a document.

Document extraction and speech synthesis

The principal technical risk is that visually similar documents do not expose equivalent machine-readable structures. Plain text can be read directly, while DOCX stores paragraphs, runs, tables, relationships, and embedded media; PDF stores positioned objects; and PPTX stores text within slide shapes. For this reason, the study compared four DOCX libraries—python-docx, docx2txt, docxpy, and docx2python [13]–[16]—and three PDF libraries—PyPDF2, PyMuPDF, and pdfminer [17]–[19]. Python-pptx was used for presentation files [20]. For speech synthesis, gTTS and pyttsx3 represented cloud-dependent and offline alternatives, respectively [21], [22].

SYSTEM DESIGN AND METHODOLOGY

Hardware and software architecture

The prototype uses a Raspberry Pi 4 as the processor, a USB pendrive as document storage, a keyboard or Braille display as the input device, and a speaker as the audio output. Python coordinates format detection, extraction, navigation state, text segmentation, user commands, and speech generation. This architecture keeps the format-specific parsers behind a common reading workflow, allowing the same Next, Previous, Enter, Escape, and Function-key combinations to be used after a file has been selected.

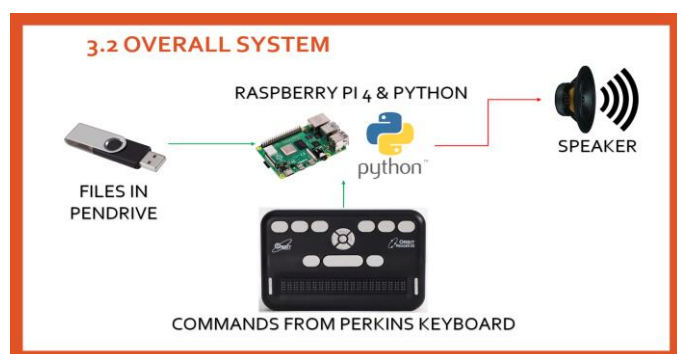


Figure 1: Overall architecture of the Raspberry Pi document-to-speech reader.

Table 1: Prototype modules and their functions.

Module	Implementation	Role in the reader
Processing	Raspberry Pi 4 and Python	Runs the interface, parsers, control logic, and audio pipeline.
Input	Keyboard or Perkins-style Braille keys	Provides navigation, commands, and six-dot Braille character entry.
Storage	USB pendrive	Supplies TXT, PDF, DOCX, and PPTX files.
Output	Speaker; optional Braille cells	Presents speech and segmented text output.

Braille-oriented control scheme

The letters S, D, F, J, K, and L are assigned to the six dots of a Braille cell. The tactile ridges on F and J provide reference positions, while the adjacent keys support stable hand placement. Space separates words; Backspace deletes a cell; Enter confirms a selection; Escape moves to the previous state; and dedicated Next and Previous inputs navigate lists, paragraphs, and slides. Function-key combinations invoke operations that would otherwise require visual menus.

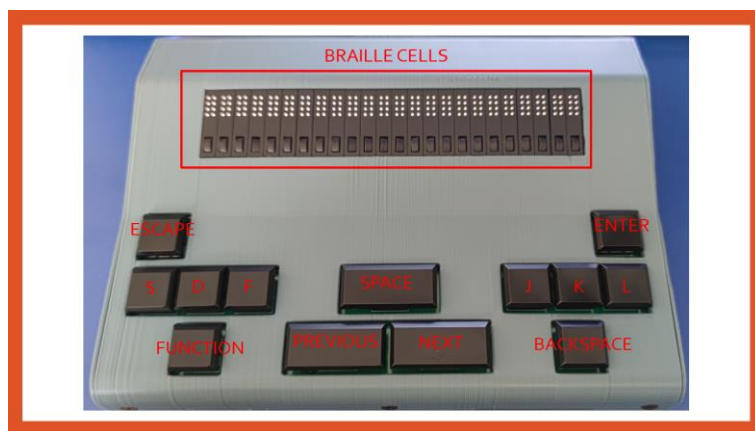


Figure 2: Prototype interface showing Braille cells and assigned command keys.

Table 2: Principal commands in the implemented interaction flow.

Operating state	Control	Action
File selection	Next / Previous	Move through files or folders.
File selection	Enter / Escape	Open or select an item / leave the folder.
Reading	Next / Previous	Speak the next or previous text segment.

Reading	Function + S	Enter a keyword and filter matching paragraphs.
File management	Function + F	Choose a new destination and relocate the selected file.

Document-processing workflow

At start-up, the user identifies a file on the USB drive. The extension determines the extraction routine: TXT is read directly, DOCX is passed to docx2python, PDF to PyPDF2, and PPTX to python-pptx. Extracted content is stored as an ordered array. The thesis implementation constrains display-oriented segments to 24 characters to match a 24-cell Braille line and limits speech requests to manageable passages, with approximately 150 words used to reduce conversion delay and remain below the observed gTTS request limit.

In reading mode, Next and Previous update the active array index and trigger speech for the selected passage. Search mode accepts a keyword through the six assigned Braille keys, filters the array to paragraphs containing the query, and then reuses the same navigation logic. Presentation mode launches LibreOffice Impress through the subprocess module; PyAutoGUI maps the tactile controls to slide-show commands. File relocation uses a directory-navigation state and confirms the new location with Enter.

Library evaluation procedure

Library selection was based on preservation of the original text and robustness across content types rather than a single aggregate accuracy percentage. Three DOCX samples were used: formatted prose containing images and hyperlinks; Greek symbols and equations; and a resume-style layout. The same representative content was evaluated in PDF form with three parsers. A PPTX sample was used to verify slide-text extraction. Outputs were compared with the visible source to identify omissions, replacement characters, ordering changes, and retained structural markers. Finally, end-to-end functions were exercised to verify that the selected components supported the required interaction flow.

RESULTS AND DISCUSSION

DOCX extraction comparison

The DOCX comparison revealed that a library capable of reading ordinary paragraphs may still fail on hyperlinks, equations, images, or layout-oriented files. In the formatted-prose sample, docx2txt and docxpy retained the full visible sentence, whereas python-docx omitted hyperlink-associated words in the recorded output. Docx2python retained the sentence and additionally exposed hyperlink targets.

In the mathematical sample, all four libraries retained the Greek characters, but only docx2python returned equation content, represented with LaTeX-like markers and some conversion errors. In the resume sample, python-docx returned no content in the test, while the other three libraries recovered the text; docx2python also reported embedded-media placeholders.

Table 3: Observed DOCX extraction outcomes across the three thesis samples.

Library	Formatted prose / links	Greek symbols / equations	Resume layout / media	Decision
python-docx	Some linked words omitted	Greek symbols only	No extracted content	Not selected
docx2txt	Visible text retained	Greek symbols only	Text retained	Not selected

docxpy	Visible text retained	Greek symbols only	Text retained	Not selected
docx2python	Text and link metadata retained	Equation markup retained with errors	Text and media markers retained	Selected

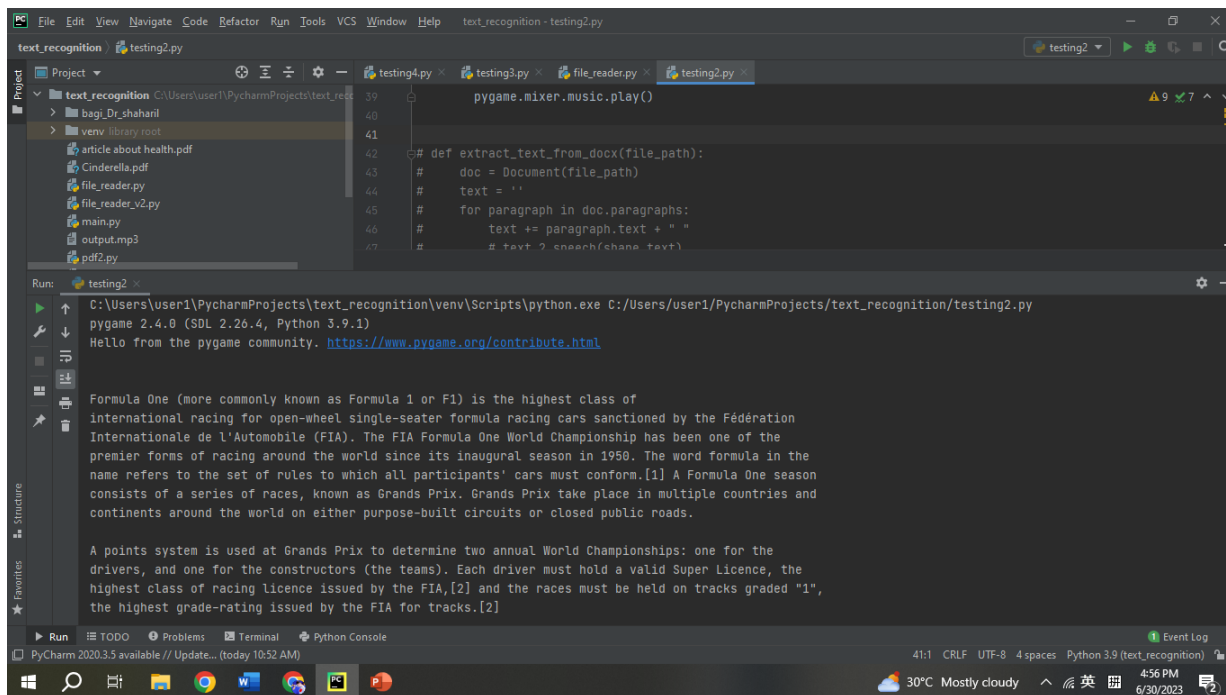


Figure 3: Example terminal output produced during document-extraction testing.

These results support docx2python for the reader because the intended task is content extraction, not document authoring. The selection does not imply perfect reconstruction: mathematical notation may be transformed, reading order can differ from the visual layout, and media markers do not describe image meaning. Nevertheless, retaining a detectable marker is preferable to silently discarding the element because the interface can alert the user that non-text content is present.

PDF and PPTX extraction

All three PDF libraries extracted the main prose sample, with minor spacing differences. The mathematical sample was more discriminating. PyPDF2 preserved the tested Greek symbols and most equation characters in the correct sequence. PyMuPDF introduced replacement characters in several positions, while pdminer altered the ordering or grouping of some fractions and operators. In the resume sample, each library recovered the principal text, although line breaks and spacing reflected the positioned nature of PDF content. PyPDF2 was therefore selected for the prototype because it gave the most faithful overall output in the recorded samples.

Table 4: Observed PDF extraction outcomes.

Library	Prose sample	Mathematical sample	Resume sample	Decision
PyPDF2	Complete; minor spacing changes	Best preservation among tested outputs	Main text recovered	Selected

PyMuPDF	Complete	Replacement characters observed	Main text recovered	Not selected
pdfminer	Complete	Some symbols reordered or grouped incorrectly	Main text recovered	Not selected

Python-pptx extracted the text from the evaluated PPTX slide without an observed textual error. Its ability to access slide shapes also supported presentation navigation and slide-specific spoken output. The implementation opens LibreOffice Impress for the visual slide show while retaining the Raspberry Pi input mapping for Next, Previous, and Escape. This division of responsibilities avoids reimplementing a presentation renderer within the reader.

Speech engine and end-to-end functionality

The comparison between gTTS and pyttsx3 exposed a direct trade-off. Pyttsx3 can operate offline and provides local control of rate, voice, and volume, but the tested implementation did not provide the required interruption behavior during ongoing speech. GTTS required an internet connection but allowed playback to be stopped within the project workflow. GTTS was therefore selected for the prototype. Pygame was used to play generated audio without maintaining a large collection of permanent audio files.

Functional verification confirmed the principal interaction paths described in Table 5. The reader restricted file selection to the supported extensions, extracted and segmented content, moved forward and backward through passages, filtered paragraphs by a Braille-entered keyword, launched and navigated a PPTX slide show, and changed a selected file's location. These observations establish implementation feasibility, but they should not be interpreted as a usability study: no timed task analysis, speech-intelligibility score, or participant-based accessibility evaluation was reported in the paper.

Table 5: End-to-end functional verification and operating conditions.

Function	Verification outcome	Important condition
Select supported files	Implemented for TXT, PDF, DOCX, and PPTX	Legacy DOC and PPT are unsupported.
Read sequentially	Next/Previous passage control implemented	Output quality depends on extraction order.
Search by keyword	Matching paragraphs can be filtered and navigated	Keyword entry uses assigned Braille keys.
Present slides	LibreOffice slide show can be launched and controlled	Available for PPTX files.
Relocate a file	Destination can be selected through the same navigation model	Requires valid storage permissions.
Generate speech	Interruptible English speech implemented with gTTS	Internet connection and processing time required.

LIMITATIONS AND IMPLICATIONS

Four limitations define the current scope. First, the reader accepts DOCX and PPTX but not the older binary DOC and PPT formats; conversion is required before use. Second, gTTS makes spoken output dependent on network availability. Third, Greek symbols and mathematical expressions can be extracted as characters or markup without being pronounced in a meaningful mathematical form. Fourth, speech generation introduces a delay that grows with passage length. Segmenting long text reduces the waiting time but can also interrupt semantic units if the split is based only on length.

The evaluation also shows why accessibility claims should be separated from parser correctness. Recovering every visible word does not guarantee a meaningful reading order, especially for multi-column layouts, equations, tables, and slides. Similarly, a small set of tactile commands reduces interface complexity but requires validation with blind users to determine discoverability, comfort, workload, and error recovery. Future work should therefore combine offline speech synthesis, structure-aware segmentation, MathML or accessible-mathematics conversion, and participant-based usability testing.

CONCLUSION

A Raspberry Pi-based reader was developed to convert TXT, PDF, DOCX, and PPTX content into spoken English through a Braille-oriented control interface. Comparative testing supported the use of docx2python for DOCX, PyPDF2 for PDF, python-pptx for presentations, and gTTS for interruptible speech in the implemented workflow. The prototype integrated file selection, sequential reading, keyword search, presentation control, and file relocation with a consistent set of tactile commands. The results demonstrate a viable portable architecture for accessing common digital learning materials without a conventional desktop workflow. They also identify clear boundaries: extraction of complex equations and layouts remains imperfect, legacy Office formats are unsupported, speech requires connectivity, and user-centred performance has not yet been quantified. Addressing these boundaries is necessary before the system can be considered a validated assistive product.

ACKNOWLEDGMENTS

The authors thank Universiti Teknologi Malaysia and the project supervisor for the facilities, technical guidance, and constructive feedback that supported this study.

REFERENCES

1. American Foundation for the Blind, "Braille Displays," 2022. Available: <https://www.afb.org/blindness-and-low-vision/using-technology/braille/braille-displays>
2. Microsoft, "Text-to-Speech," 2021. Available: <https://www.microsoft.com/en-us/accessibility/products/windows>
3. Freedom Scientific, "Refreshable Braille Displays," 2022. Available: <https://www.freedomscientific.com/products/blindness/braille-displays/>
4. K. Kiyota, T. Ishibashi, M. Shimakawa, and K. Ito, "Effects of social implementation education for assistive device engineers at NIT (KOSEN) through the development of a digital reading device for the visually impaired," *Sensors*, vol. 22, no. 3, p. 1047, 2022.
5. NV Access, "NVDA," 2022. Available: <https://www.nvaccess.org/download/>
6. Freedom Scientific, "JAWS," 2022. Available: <https://www.freedomscientific.com/products/software/jaws/>
7. Dolphin Computer Access, "Dolphin Screen Reader," 2022. Available: <https://yourdolphin.com/en-gb/products/individuals/dolphin-screen-reader>
8. Apple, "VoiceOver," 2022. Available: <https://www.apple.com/accessibility/mac/vision/>
9. BRLTTY Project, "BRLTTY," 2022. Available: <https://mielke.cc/brlTTY/>
10. V. Phutak, R. Kamble, S. Gore, M. Alave, and R. R. Kulkarni, "Text to speech conversion using Raspberry-Pi," *International Journal of Innovative Science and Research Technology*, vol. 4, no. 2, 2019.

11. T. Shah and S. Parshionikar, "Efficient portable camera based text to speech converter for blind person," in 2019 International Conference on Intelligent Sustainable Systems (ICISS), 2019, pp. 353–358.
12. S. Salvi, S. Pahar, and Y. Kadale, "Smart glass using IoT and machine learning technologies to aid the blind, dumb and deaf," Journal of Physics: Conference Series, vol. 1804, no. 1, p. 012181, 2021.
13. python-docx, 2022. Available: <https://python-docx.readthedocs.io/>
14. docx2txt, 2022. Available: <https://pypi.org/project/docx2txt/>
15. docxpy, 2022. Available: <https://pypi.org/project/docxpy/>
16. docx2python, 2022. Available: <https://pypi.org/project/docx2python/>
17. PyPDF2, 2022. Available: <https://pypi.org/project/PyPDF2/>
18. PyMuPDF, 2022. Available: <https://pypi.org/project/PyMuPDF/>
19. pdfminer, 2022. Available: <https://pypi.org/project/pdfminer/>
20. python-pptx, 2022. Available: <https://python-pptx.readthedocs.io/>
21. gTTS, 2022. Available: <https://pypi.org/project/gTTS/>
22. pyttsx3, 2022. Available: <https://pypi.org/project/pyttsx3/>