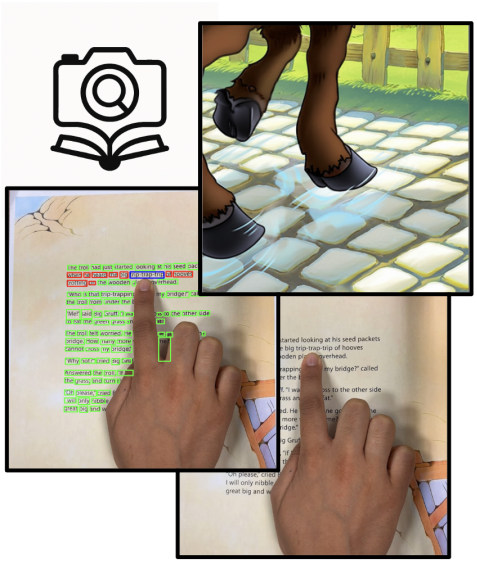
LEXICAM: A VOICE-ACTIVATED, VISION-GUIDED REAL-TIME READING COMPANION

Saniya Karwa, Asmi Kumar, & Khushi Parikh *
{saniya,asmi,khushi25}@mit.edu
6.8510 Intelligent Multimodal User Interfaces
Spring 2025



ABSTRACT

Lexicam is an interactive, camera-guided reading assistant designed to help children independently learn the meanings of unfamiliar words while reading physical books. The system combines computer vision, natural language processing (NLP), and text-to-speech technology to provide real-time vocabulary support. Using a smartphone camera, Lexicam detects the user's pointing finger through Google's MediaPipe hand landmarking model and captures an image of the word being pointed to. Python's `pytesseract` is employed for optical character recognition (OCR) to extract text from the page, and the system utilizes Gemini-2.0-Flash to generate simple, kid-friendly definitions and accompanying images of the word. The definition is then read aloud to the user using `pyttsx3`. Lexicam operates effectively with picture books, although challenges with more complex layouts, such as in graphic novels or dense chapter books, were observed. Despite issues with hand detection accuracy and OCR reliability, the system delivers useful and contextually relevant word definitions, with the added benefit of visual aids.

* All authors have contributed equally.

1 Introduction and Overview

Reading is one of the most fundamental skills in a child's early development, but many children struggle with independently understanding unfamiliar vocabulary. While digital reading tools exist, most are optimized for screen-based reading, leaving a gap for children who read from physical books. When a child encounters an unfamiliar word in print, they often rely on an adult to provide context or definitions, but such support isn't always available.

Lexicam addresses this gap by allowing children to read physical books independently while still receiving vocabulary support. By combining camera-based word recognition with voice-activated queries, Lexicam lets a child simply point to a word in a book and ask, "Hey Lexi, what does this mean?" In response, the system provides an age-appropriate definition and a helpful image, all without requiring them to leave the page or navigate complex menus.

Existing tools like Google and voice assistants aren't built for young readers. Typing requires spelling, voice assistants expect correct pronunciation, and using context clues can quickly become frustrating with unfamiliar words. Lexicam removes those hurdles. It lets children learn in the flow of reading—no typing, no guessing how to say the word—just point and ask. By making vocabulary support effortless and intuitive, Lexicam helps kids stay focused and explore independently.

This project is interesting because it combines multiple modalities, computer vision, speech recognition, and natural language processing, to create a seamless and intuitive learning experience. It empowers young readers to ask questions naturally and get immediate answers, promoting autonomy, curiosity, and confidence in reading.

2 System Description

Lexicam enables real-time, voice-guided word assistance while a child reads from a physical book. It monitors the book through a live video feed from a camera and listens for a voice trigger, such as "Hey Lexi." When activated, the system captures a snapshot of the current page, detects the location of the user's pointing finger, and identifies the word at that location using OCR and spatial analysis. It then looks up the meaning of the word in context, and provides a simple, child-friendly definition along with a supporting image. To enhance accessibility and learning, Lexicam also reads the definition aloud using text-to-speech, reinforcing comprehension for early readers.

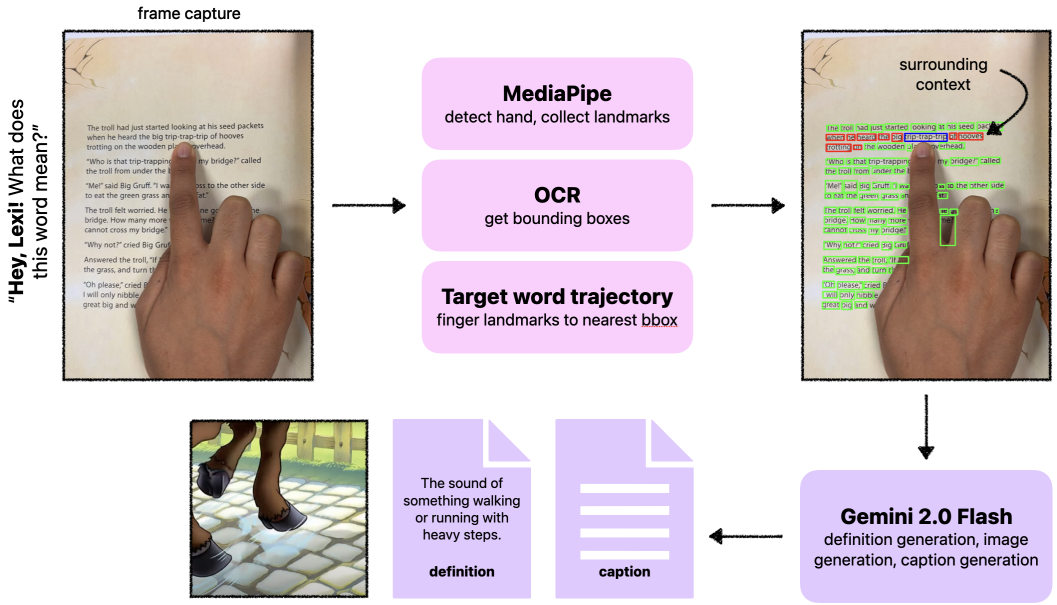


Figure 1: End-to-end example of an example of our pipeline.

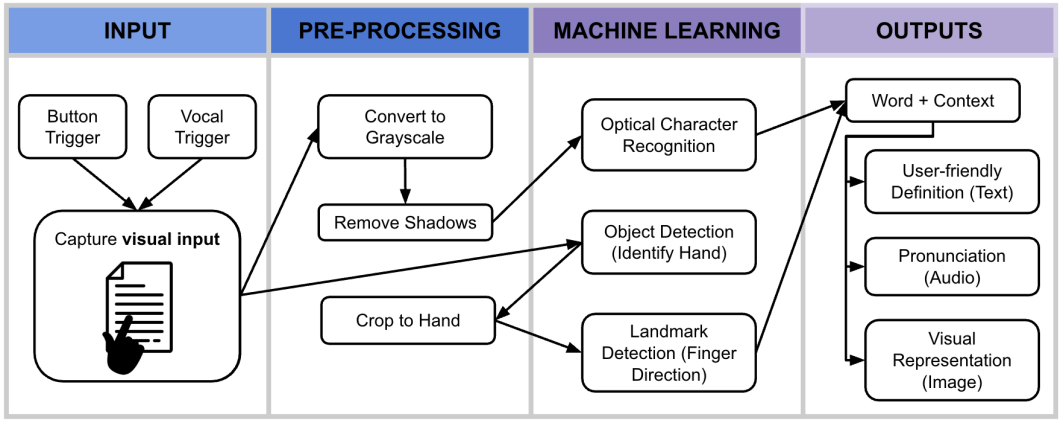


Figure 2: Full technical sequence.

3 How It Works

Lexicam's real-time assistance emerges from the integration of three technical stages: precise hand-landmark detection, robust OCR and pointing logic, and context-aware generative AI output. These stages execute in sequence on each triggered frame, smoothly transforming a user's gesture into an instant, multimodal explanation. Together, these are noted in Figure 2.

3.1 Audio Trigger and Hand Landmark Capture

The system continuously listens on a background thread for the wake word "lex" using Python's `speech_recognition` library. We chose the word "lex" to accommodate various possible spellings of "Lexi" that may arise during speech-to-text translation. When the user says "Hey, Lexi," the thread sets a shared `triggered` flag. Meanwhile, in the main loop running a live 1280×720 video feed from the camera, each frame is asynchronously passed to MediaPipe's HandLandmarker, configured with low hand detection sensitivity thresholds to improve robustness under varied lighting and background clutter. As soon as the audio trigger is detected, the current frame is saved to disk, and the most recent hand landmarks are extracted. Specifically, we capture two keypoints on the pointing hand: the proximal interphalangeal (pip) joint, and the fingertip (tip) joint.

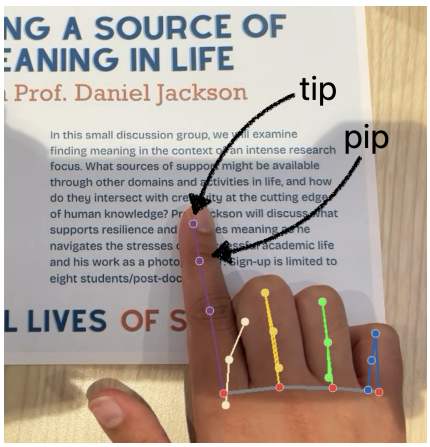


Figure 3: Tip and pip landmarks are the top two coordinates on the pointer finger

These normalized coordinates are scaled to pixel space and saved as (x, y) pairs. This precise pairing of speech trigger and landmark capture ensures that the exact moment of pointing is preserved for downstream OCR.

3.2 OCR PREPROCESSING AND TARGET WORD DETECTION

Once the frame and landmark data have been saved, the next step is to identify which word the user is pointing to. This involves two key tasks: cleaning up the image for optical character recognition (OCR) and intersecting the computed pointing trajectory with recognized text bounding boxes.

To prepare the image for OCR, Lexicam applies a custom sequence of image preprocessing steps. The input frame is first converted to grayscale. This grayscale image is then dilated with a 7×7 kernel to reduce small-scale noise and sharpen character boundaries. The result is used to compute a background estimate via a median blur. Then, we can amplify the visibility of foreground text while suppressing shadows and uneven lighting. The contrast-enhanced image is then normalized and binarized, yielding a crisp image optimized for text recognition.

This preprocessed image is passed to Python's `pytesseract` OCR library, which extracts individual words along with their bounding boxes and positional indices. Lexicam filters out all empty or whitespace-only detections and constructs a list of words. Each word is associated with a bounding box (x_1, y_1, x_2, y_2), representing the top-left and bottom-right corners of the word in image coordinates.

Next, Lexicam determines the pointing direction by loading the tip and pip joint coordinates from the previous step. These two points are interpreted as the start and direction vector of a ray, with the ray defined as

$$\vec{r}(t) = \mathbf{x}_{\text{tip}} + t \cdot (\mathbf{x}_{\text{tip}} - \mathbf{x}_{\text{pip}}), \quad t > 0,$$

which extends upwards past the fingertip in the direction the user's finger is pointing. Lexicam finds the closest bounding box, and thus, associated word, where the ray intersects one of its edges.

Once such a target word is identified, a symmetric context window of up to 10 words before and after the target is collected to construct a sentence-level context string. The frame is annotated in Figure 1 with colored rectangles: blue for the target word, red for context words, and green for all other recognized words. These annotations, along with the target and context text, are saved for the final stage of generative output.

3.3 CONTEXT-AWARE GENERATIVE AI OUTPUT

With the target word and its surrounding context extracted, Lexicam proceeds to generate an appropriate definition and visual illustration using a multimodal large language model. This final stage is handled by a generative AI pipeline powered by Google's Gemini 2.0 Flash API, which returns both natural language and image outputs tailored to the user's reading moment.

To begin, we constructed a natural-language prompt by embedding the target word and its local sentence context into a predefined instruction template. This prompt instructs the model with the following prompt:

```
You'll be given a word, along with the context around
the word. Provide a simple, kid-friendly definition
of the word in the given context. Your output should be
in JSON format:
{'word': <word>, 'definition': <your definition>}

The word is: {target['target_word']}.
The context is: {target['context']}.
```

This prompt is sent to the Gemini 2.0 Flash text generation endpoint. The response is parsed to extract the word and definition.

Next, Lexicam requests a visual illustration by issuing a second prompt to the Gemini image generation endpoint. This prompt is similarly structured:

```
You'll be given a word, along with the context around
the word. Generate a descriptive, kid-friendly image that
best depicts the word, in the context provided.
```

```
Your output should be AN IMAGE. GENERATE an image.

The word is: {target['target_word']}.
The context is: {target['context']}.
```

The model responds with both a generated image and a detailed image caption.

Finally, Lexicam uses the `pyttsx3` text-to-speech engine to convert the definition into audio. The engine is configured with a human-like voice and a slightly slowed speaking rate for clarity and accessibility. The audio playback occurs in sync with the display of the definition and generated image, creating a seamless, multimodal learning experience.

4 Evaluation

Lexicam's performance varied across book types, use cases, and physical environments. While the system excelled in many scenarios, it also revealed a range of limitations. Many of these led to meaningful design pivots and improvements.

4.1 Performance Analysis and Observed Behaviors

We tested Lexicam with a variety of physical books, ranging from picture books to chapter books and graphic novels. Across these formats, we observed that the system performed best with picture books and introductory chapter books. These books typically featured large, clearly separated text blocks and minimal background clutter, which aligned well with our OCR pipeline and hand-tracking calibration and thresholds. The layouts made it easier for the system to extract bounding boxes and accurately identify target words from pointing gestures. Fortunately, this also aligns closely with our target audience of young children.

However, performance degraded on different types of page layouts. Graphic novels posed a particular challenge due to their dense formatting, presence of speech bubbles, overlay of text on illustrated backgrounds, and use of non-standard fonts. These elements confused both the OCR and hand-tracking systems. In such cases, MediaPipe often failed to detect a hand, and even when it did, the OCR struggled to segment and recognize meaningful text.

Chapter books, though visually simpler, introduced other problems. The narrow line spacing, small font sizes, and fixed top-down camera height meant that it was hard to focus on the text, especially near the bottom of the frame. These issues made it difficult to maintain sharpness across the entire reading area and caused OCR accuracy to decline. Based on these observations, we propose that a dynamic zoom mechanism focused around the pointing ray, or use of a high-resolution, auto-focusing camera, could meaningfully improve text clarity.

In terms of system latency, we observed an average delay of 5–6 seconds between the trigger phrase ("Hey Lexi") and the start of the spoken output and image pop-up. This response time is short enough to feel natural for children, maintaining the flow of interaction and keeping users engaged while waiting.

Importantly, Lexicam handled both standard vocabulary and invented or unusual words commonly found in children's literature (e.g., "troll," "trip-trap-trip," "alluring"). Gemini 2.0 Flash proved adept at generating context-aware definitions and relevant illustrations even for nonsensical or fictional words, demonstrating the flexibility and power of the model in handling real-world examples beyond dictionary-based approaches.

4.2 An Interesting Failure

One particularly instructive failure occurred in the ray-casting logic used to determine the intended target word. Our implementation uses only two hand landmarks—the tip joint and the pip joint—to cast a directional ray from the finger into the image. In some test cases, small variations in landmark detection within MediaPipe (e.g., one/both landmark(s) slightly misaligned) could cause the ray to miss the intended word and instead intersect a neighboring bounding box. This resulted in the system targeting the wrong word.

This issue occurred when the finger was not fully extended (and all the joints were not visible) or when the user's hand was tilted at an extreme angle. Because we rely on just two points, minor errors in those positions can drastically alter the projected trajectory. A promising future improvement would be to use additional joints further along the pointing finger and average multiple trajectories, reducing sensitivity to local jitter and yielding more robust pointing detection.

4.3 Implementation Challenges and Design Pivots

Several components of Lexicam proved more difficult to implement than expected. The first major hurdle was OCR accuracy. While we initially expected off-the-shelf Tesseract to be sufficient, we quickly discovered that real-world reading conditions—such as shadows cast by the user's hand, partially occluded letters, and varied lighting—led to poor OCR output. Shadows in particular were often misinterpreted as text regions. To address this, we introduced a preprocessing pipeline that included grayscale conversion, background subtraction using a median blur, and adaptive thresholding to isolate foreground text. This improved performance significantly, though the OCR is still somewhat fragile to partial word occlusion.

Hand detection was another area that required considerable tuning. Against cluttered or dimly lit environments, MediaPipe often failed to register hands at all. After experimenting with detection thresholds and adjusting confidence settings, we found that lowering the threshold for hand detection (accepting more false positives) improved reliability. However, this approach still required careful calibration to avoid much noise during hand-tracking.

On the language model front, we initially planned to combine a dictionary API with a smaller, cheap LLM (e.g., GPT-2) to rank definitions based on context. However, GPT-2 struggled with contextual disambiguation, especially when words had multiple meanings or required nuanced interpretation. This led us to pivot to Gemini 2.0 Flash, which has a free API and offered integrated text and image generation and performed far better on both real and invented vocabulary. Adopting Gemini significantly simplified our system and improved both accuracy and user experience.

4.4 What Worked Easily

While some subsystems demanded substantial engineering effort, others worked surprisingly well out of the box. Gemini 2.0 Flash consistently provided clear, context-sensitive definitions and generated relevant, engaging illustrations. It handled both common and rare vocabulary gracefully, including nonsense words that would not appear in any standard dictionary. The ability to use a single prompt to produce a definition, an image, and a caption streamlined our architecture and reduced the need for separate pipelines or fallback logic. This reliability made Gemini a cornerstone of the Lexicam experience and validated our choice to center the system around multimodal AI output.

5 User Study

We tested Lexicam with three of our classmates to see how well it worked in the hands of new users. Each person was given a children's book and asked to use the system to look up unfamiliar or interesting words. After a brief demo, all participants were able to trigger Lexicam and point at words without help, confirming that the voice-and-gesture interface felt intuitive and easy to learn.

That said, the study surfaced a few issues. One participant often pointed too close to the word, accidentally covering it with their finger. As a result, OCR failed or returned the wrong word. Another user said the trigger phrase too softly, so it didn't register, which is understandable. These moments showed us the need for clearer feedback; we implemented a visual cue to let the user know when the system is successfully triggered by pausing the live video feed. Though not currently implemented, it would be helpful to push a warning if the finger is blocking text.

Interestingly, one user tried to select a full phrase like "trip-trap-trip" by swiping their finger across it. Although Lexicam handled this correctly in that case, it made us think more about supporting multi-word selection intentionally.

Overall, the study confirmed Lexicam's core interaction flow works well for new users, but also highlighted the need for small improvements in feedback and gesture handling that we might have missed without seeing real users in action.

6 Future Work and Capabilities

An important capability just out of reach for Lexicam is contextual understanding at the level of the full book. Currently, our system only sees and processes the visible content at the time the user points. This limits Lexicam's ability to answer questions about the broader story or to reason beyond sentence-level context. A natural next step would be to detect when the page has changed—either by detecting a literal page flip or more simply when the visible content has been updated—and proactively cache the full text of each page as it appears.

One way to achieve this would be to run lightweight OCR every few seconds and compare the first few words of the page. If those words differ significantly from the previous scan, we could infer a page change and reprocess the new content in full. This cached text could then serve as context for future queries, enabling features like question-answering about characters, summarization, or story-based clarification. Users would still be able to point to a word and receive a definition, but that definition could now be grounded in the full narrative arc of the story.

Another promising extension would be multi-modal alignment between images and text—especially relevant for illustrated books. If Lexicam could associate visual elements on the page (e.g., an image of a spider) with text nearby ("Charlotte spun a web"), it could answer questions like "Who is this?" or "What is happening here?" when the user points to an image. This would broaden Lexicam's utility from vocabulary support to general reading comprehension and storytelling aid.

Appendix

Here are Lexicam's top-level dependencies. A full list is provided here.

Package	Version	Usage
`google-genai`	$\geq$ 1.13.0	Generates context-aware definitions, images, and captions using Gemini 2.0 Flash
`ipykernel`	$\geq$ 6.29.5	Enables running Lexicam in a Jupyter notebook environment
`mediapipe`	$\geq$ 0.10.21	Performs real-time hand tracking and landmark detection
`opencv-python`	$\geq$ 4.11.0.86	Handles image processing for OCR and drawing bounding boxes
`pyttsx3`	$\geq$ 2.98	Converts text into spoken audio using the system's built-in TTS engine
`pyaudio`	$\geq$ 0.2.14	Records or plays raw audio streams from microphones and speakers
`pytesseract`	$\geq$ 0.3.13	Extracts text from images via Tesseract OCR
`speechrecognition`	$\geq$ 3.14.2	Detects the spoken trigger phrase to activate Lexicam

Our complete code is provided on GitHub.