Department of Computer Science & Information Technology

CT-361: Artificial Intelligence & Expert Systems

PROJECT REPORT

“SmartSage”

GROUP MEMBERS:

- YUMNA IRFAN (CT-22004)
- ALEESHBA (CT-22005)
- HADIYA KASHIF (CT-22008)
- YASHA ALI (CT-22010)
- SARA RAZEEN (CT-22049)

Contents

1. Introduction

The growing complexity and volume of digital information present significant challenges in extracting relevant knowledge efficiently. Traditional search systems return a list of links rather than meaningful, personalized insights, and most assistants fail to combine retrieval and generation robustly. To bridge this gap, we developed an AI-powered assistant that integrates Retrieval-Augmented Generation (RAG), multimodal interaction capabilities (text and voice), and productivity-enhancing features like note creation, flashcard generation, webcam based fatigue detection along with Pomodoro timer and task management and web data integration. Built using Flask for accessibility and rapid development, the system focuses on improving how users' access, digest, and utilize information across formats.

2. Problem Statement

Most users—whether students, researchers, or professionals—face challenges in retrieving precise and contextually relevant information from large documents or scattered web sources. Moreover, learning and retention often require structured notes or summaries, which are time-consuming to prepare. Existing tools also lack flexible interaction modes (like voice input/output) and contextual understanding needed for deeper questions. People tend to lose focus quickly and often work is delayed due to this. So, the project addresses these problems by designing a system that acts as a smart, multimodal knowledge companion.

3. Objectives

- To implement a Retrieval-Augmented Generation (RAG) system capable of processing and generating answers from any textual document.
- To incorporate advanced AI techniques such as multi-query generation, query decomposition, and RAG fusion for enhanced response relevance and depth.
- To enable multimodal interaction, allowing users to communicate through both speech and text.
- To support active learning by generating structured notes and flashcards from responses.
- To allow users to export chat sessions in various formats (e.g., TXT, PDF, JSON) for review and documentation.
- To enhance information accuracy by integrating real-time data using web scraping capabilities.
- To promote productivity and wellness through a Pomodoro-based task manager, fatigue detection via webcam, and personalized motivational feedback.
- To track task-specific fatigue and customize motivational content based on task type and user condition.
- To support Memory Anchoring by generating concise summaries, mnemonics, and contextual examples for custom terms to reinforce retention.

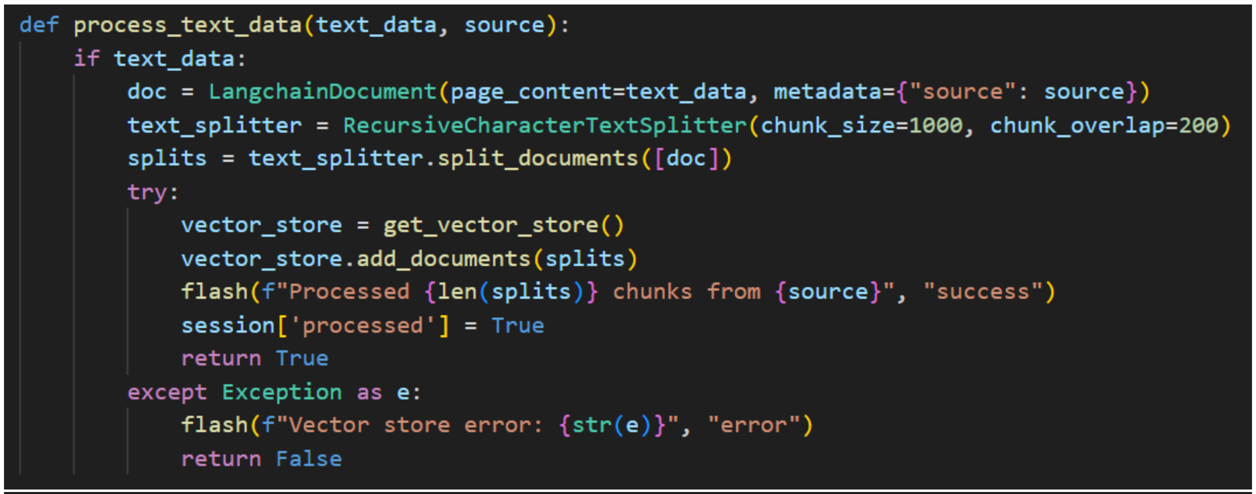
4. System Design & Feature Justification

The system is designed as a modular, intelligent AI assistant that enhances user productivity and learning through integrated retrieval, generation, and interaction features. Each module not only performs a defined function but also brings AI-enhanced capabilities to deliver a seamless and enriched user experience.

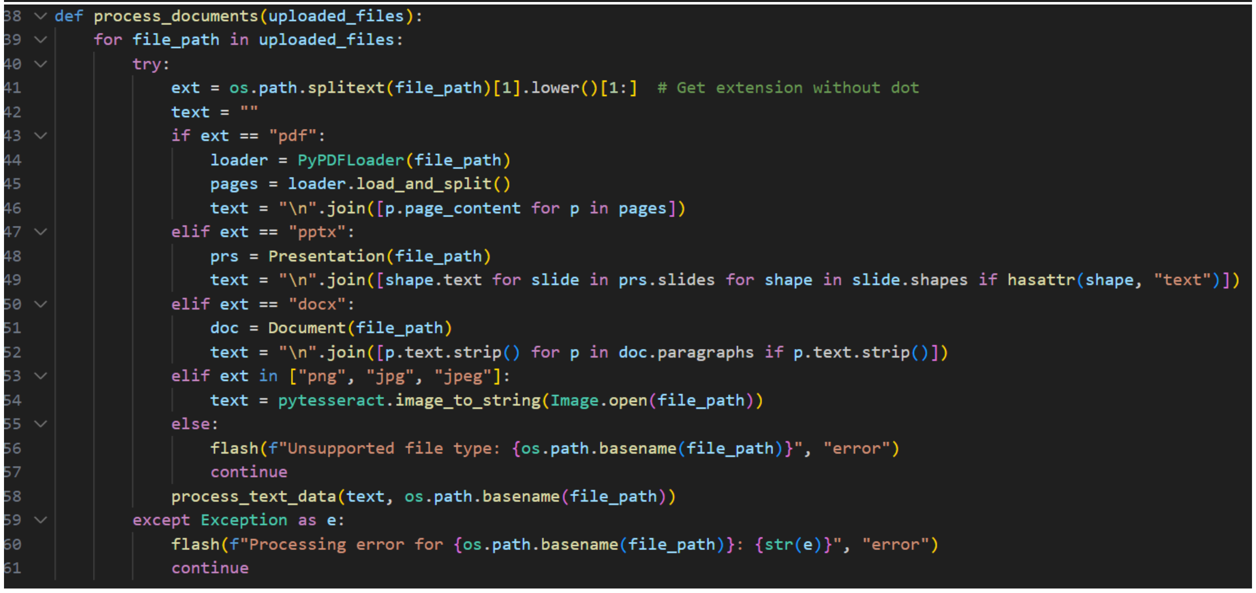
1. Input Handler

Function: Allows users to upload files in various formats including PDF, DOCX, PPTX, JPG, and PNG.

Justification: Supports multimodal content intake to ensure flexibility and ease of access to user data sources.



```python
def process_text_data(text_data, source):
    if text_data:
        doc = LangchainDocument(page_content=text_data, metadata={"source": source})
        text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
        splits = text_splitter.split_documents([doc])
        try:
            vector_store = get_vector_store()
            vector_store.add_documents(splits)
            flash(f"Processed {len(splits)} chunks from {source}", "success")
            session['processed'] = True
            return True
        except Exception as e:
            flash(f"Vector store error: {str(e)}", "error")
            return False
```

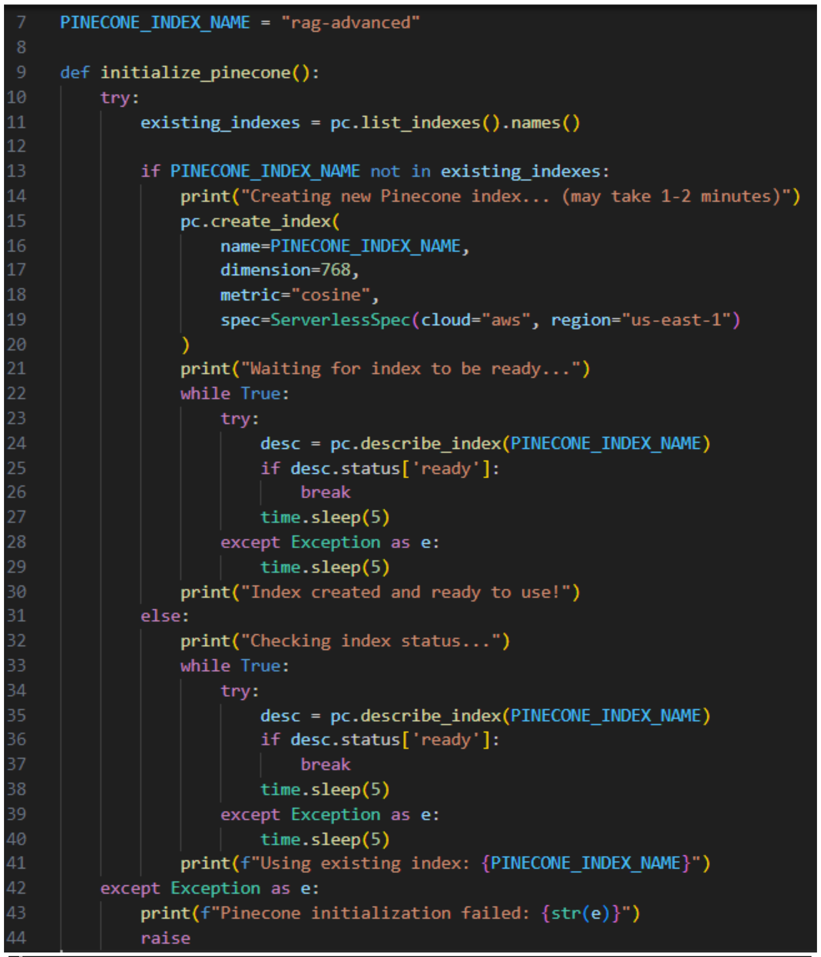


```python
def process_documents(uploaded_files):
    for file_path in uploaded_files:
        try:
            ext = os.path.splitext(file_path)[1].lower()[1:]  # Get extension without dot
            text = ""
            if ext == "pdf":
                loader = PyPDFLoader(file_path)
                pages = loader.load_and_split()
                text = "\n".join([p.page_content for p in pages])
            elif ext == "pptx":
                prs = Presentation(file_path)
                text = "\n".join([shape.text for slide in prs.slides for shape in slide.shapes if hasattr(shape, "text")])
            elif ext == "docx":
                doc = Document(file_path)
                text = "\n".join([p.text.strip() for p in doc.paragraphs if p.text.strip()])
            elif ext in ["png", "jpg", "jpeg"]:
                text = pytesseract.image_to_string(Image.open(file_path))
            else:
                flash(f"Unsupported file type: {os.path.basename(file_path)}", "error")
                continue
            process_text_data(text, os.path.basename(file_path))
        except Exception as e:
            flash(f"Processing error for {os.path.basename(file_path)}: {str(e)}", "error")
            continue
```

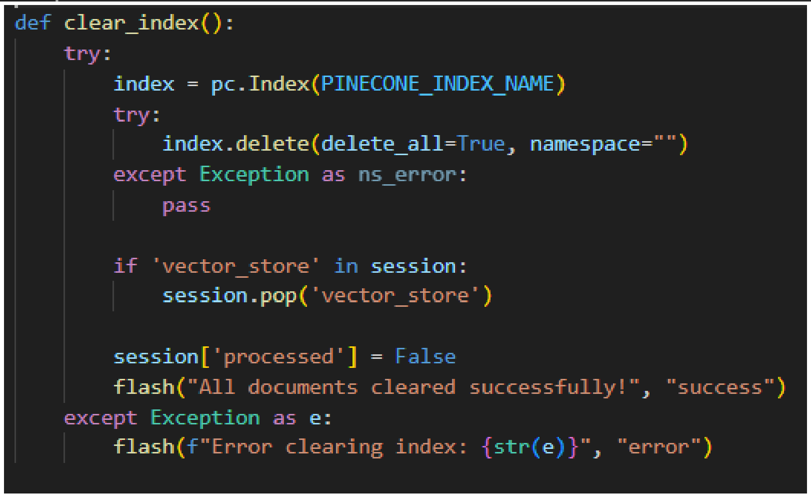
2. Vector Database (Pinecone)

Function: Uploaded content is chunked and embedded. The resulting vector representations are stored in Pinecone.

Justification: Enables scalable, high-performance semantic search for retrieving only the most relevant segments. Forms the foundation of accurate document-grounded response generation.



```python
PINECONE_INDEX_NAME = "rag-advanced"

def initialize_pinecone():
    try:
        existing_indexes = pc.list_indexes().names()

        if PINECONE_INDEX_NAME not in existing_indexes:
            print("Creating new Pinecone index... (may take 1-2 minutes)")
            pc.create_index(
                name=PINECONE_INDEX_NAME,
                dimension=768,
                metric="cosine",
                spec=ServerlessSpec(cloud="aws", region="us-east-1")
            )
            print("Waiting for index to be ready...")
            while True:
                try:
                    desc = pc.describe_index(PINECONE_INDEX_NAME)
                    if desc.status['ready']:
                        break
                    time.sleep(5)
                except Exception as e:
                    time.sleep(5)
            print("Index created and ready to use!")
        else:
            print("Checking index status...")
            while True:
                try:
                    desc = pc.describe_index(PINECONE_INDEX_NAME)
                    if desc.status['ready']:
                        break
                    time.sleep(5)
                except Exception as e:
                    time.sleep(5)
            print(f"Using existing index: {PINECONE_INDEX_NAME}")
    except Exception as e:
        print(f"Pinecone initialization failed: {str(e)}")
        raise
```



```python
def clear_index():
    try:
        index = pc.Index(PINECONE_INDEX_NAME)
        try:
            index.delete(delete_all=True, namespace="")
        except Exception as ns_error:
            pass

        if 'vector_store' in session:
            session.pop('vector_store')

        session['processed'] = False
        flash("All documents cleared successfully!", "success")
    except Exception as e:
        flash(f"Error clearing index: {str(e)}", "error")
```

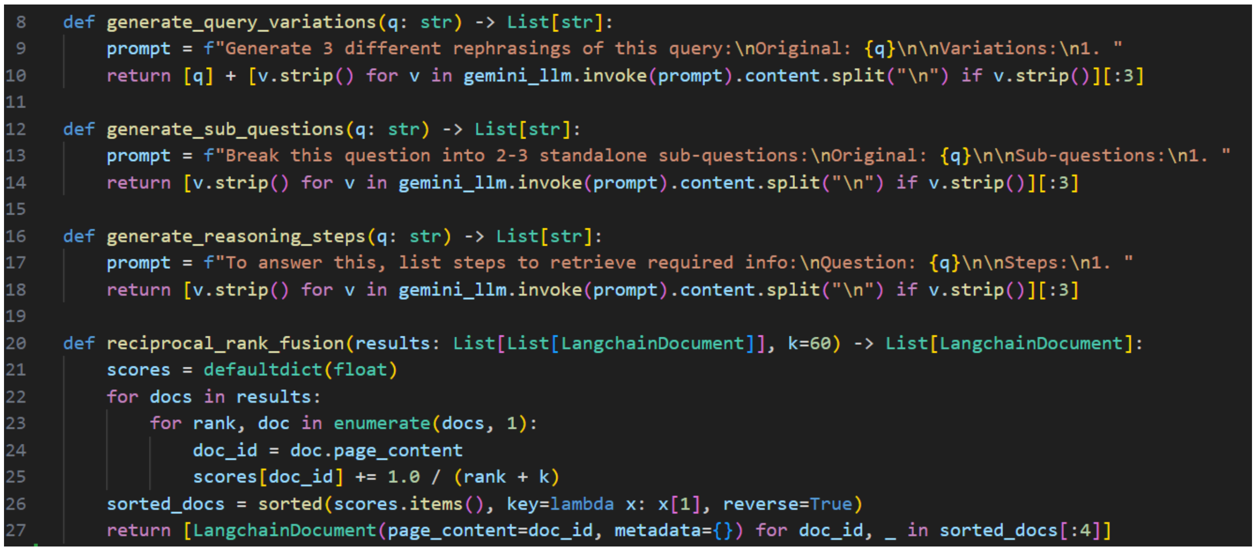
3. Query Engine with RAG, Multi-query, Decomposition, and Fusion

Function: Enhances user input through:

- Multi-query generation: Reformulates a single prompt into multiple semantically varied queries.
- Query decomposition: Breaks complex queries into sub-questions.
- RAG fusion: Synthesizes responses from multiple retrieved chunks into a coherent answer.

Powered by Gemini 2.0 Flash API for natural language understanding and generation.

Justification: This module ensures broad document coverage and deep reasoning, leading to more accurate and contextually enriched responses.

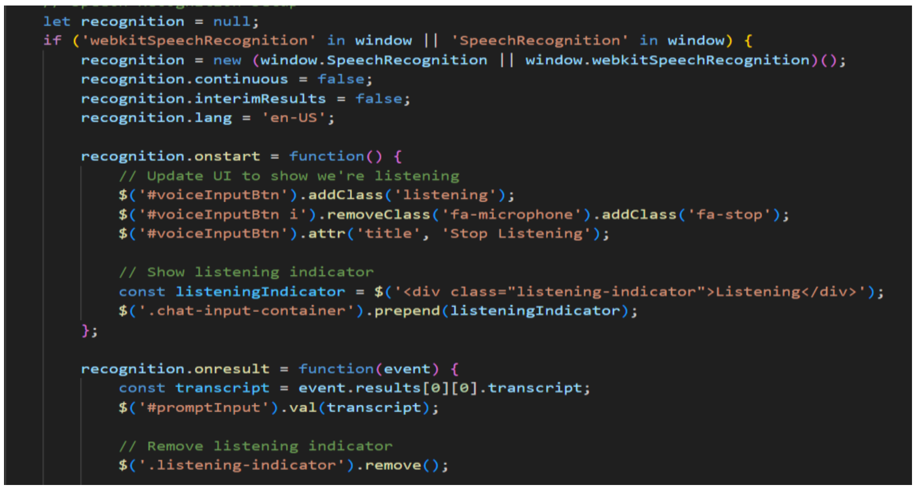


```python
def generate_query_variations(q: str) -> List[str]:
    prompt = f"Generate 3 different rephrasings of this query:\nOriginal: {q}\n\nVariations:\n1. "
    return [q] + [v.strip() for v in gemini_llm.invoke(prompt).content.split("\n") if v.strip()][:3]

def generate_sub_questions(q: str) -> List[str]:
    prompt = f"Break this question into 2-3 standalone sub-questions:\nOriginal: {q}\n\nSub-questions:\n1. "
    return [v.strip() for v in gemini_llm.invoke(prompt).content.split("\n") if v.strip()][:3]

def generate_reasoning_steps(q: str) -> List[str]:
    prompt = f"To answer this, list steps to retrieve required info:\nQuestion: {q}\n\nSteps:\n1. "
    return [v.strip() for v in gemini_llm.invoke(prompt).content.split("\n") if v.strip()][:3]

def reciprocal_rank_fusion(results: List[List[LangchainDocument]], k=60) -> List[LangchainDocument]:
    scores = defaultdict(float)
    for docs in results:
        for rank, doc in enumerate(docs, 1):
            doc_id = doc.page_content
            scores[doc_id] += 1.0 / (rank + k)
    sorted_docs = sorted(scores.items(), key=lambda x: x[1], reverse=True)
    return [LangchainDocument(page_content=doc_id, metadata={}) for doc_id, _ in sorted_docs[:4]]
```

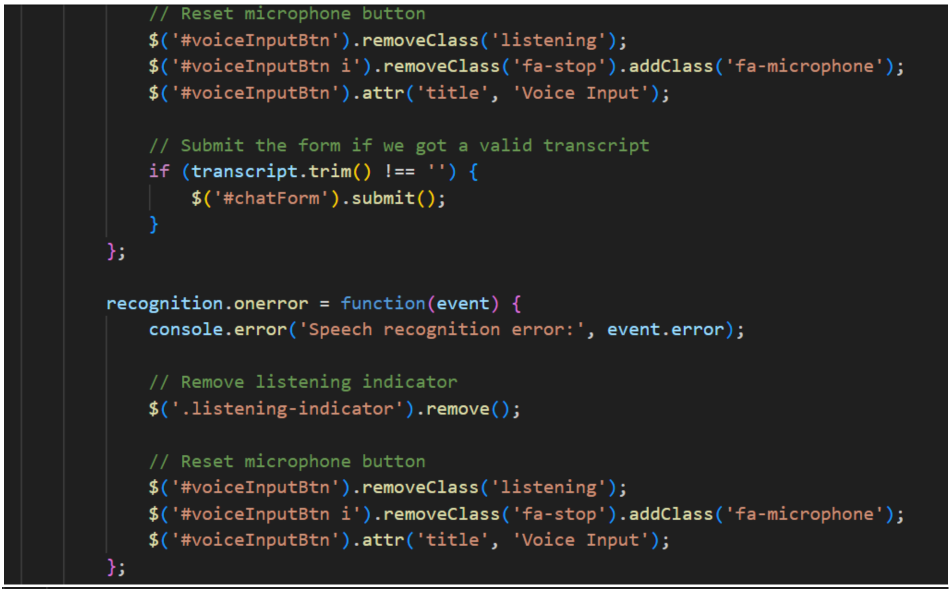
4. Voice Interaction Module (Speech-to-Text & Text-to-Speech)

Function: Converts voice input into text and speaks the assistant's response.

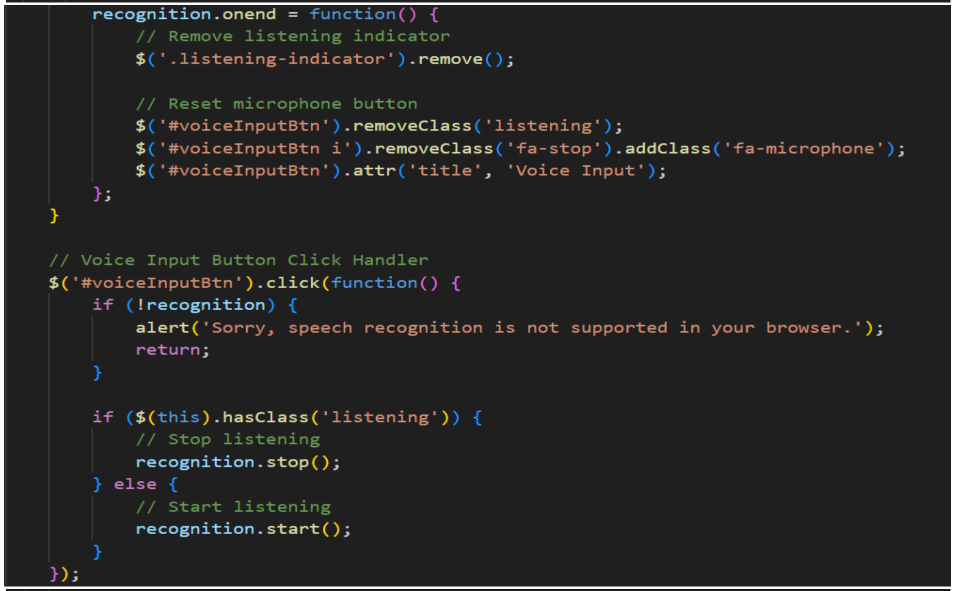
Justification: Improves accessibility, enables hands-free interaction, and supports learning-on-the-go and multitasking.



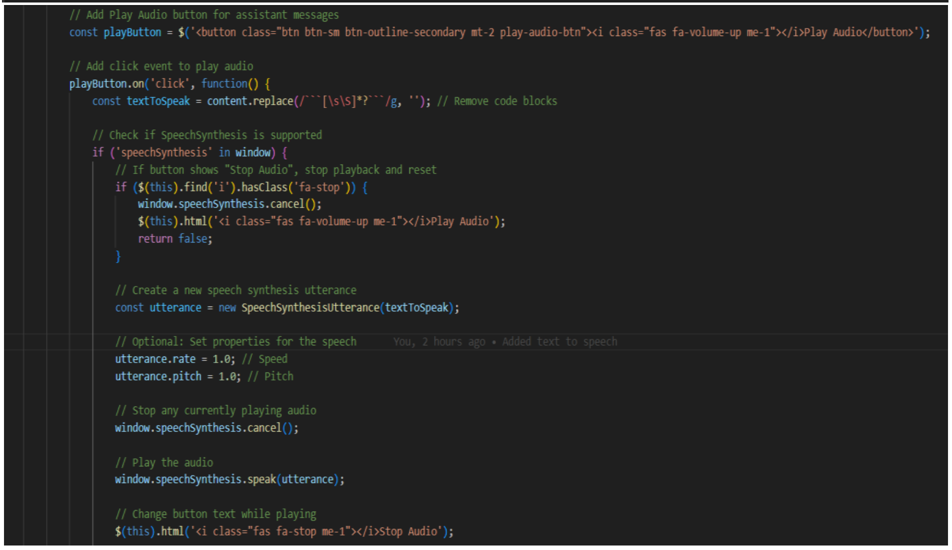
```javascript
let recognition = null;
if ('webkitSpeechRecognition' in window || 'SpeechRecognition' in window) {
    recognition = new (window.SpeechRecognition || window.webkitSpeechRecognition)();
    recognition.continuous = false;
    recognition.interimResults = false;
    recognition.lang = 'en-US';

    recognition.onstart = function() {
        // Update UI to show we're listening
        $('#voiceInputBtn').addClass('listening');
        $('#voiceInputBtn i').removeClass('fa-microphone').addClass('fa-stop');
        $('#voiceInputBtn').attr('title', 'Stop Listening');

        // Show listening indicator
        const listeningIndicator = $('<div class="listening-indicator">Listening</div>');
        $('.chat-input-container').prepend(listeningIndicator);
    };

    recognition.onresult = function(event) {
        const transcript = event.results[0][0].transcript;
        $('#promptInput').val(transcript);

        // Remove listening indicator
        $('.listening-indicator').remove();
```



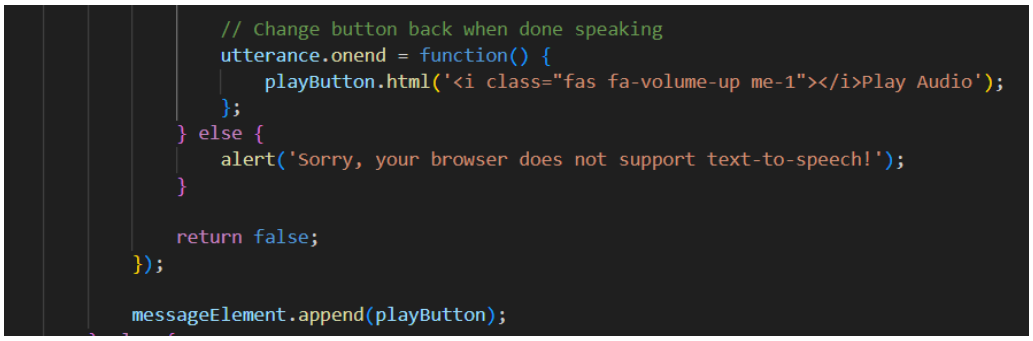
```
            // Reset microphone button
            $('#voiceInputBtn').removeClass('listening');
            $('#voiceInputBtn i').removeClass('fa-stop').addClass('fa-microphone');
            $('#voiceInputBtn').attr('title', 'Voice Input');

            // Submit the form if we got a valid transcript
            if (transcript.trim() !== '') {
                $('#chatForm').submit();
            }
        };

        recognition.onerror = function(event) {
            console.error('Speech recognition error:', event.error);

            // Remove listening indicator
            $('.listening-indicator').remove();

            // Reset microphone button
            $('#voiceInputBtn').removeClass('listening');
            $('#voiceInputBtn i').removeClass('fa-stop').addClass('fa-microphone');
            $('#voiceInputBtn').attr('title', 'Voice Input');
        };
```



```
        recognition.onend = function() {
            // Remove listening indicator
            $('.listening-indicator').remove();

            // Reset microphone button
            $('#voiceInputBtn').removeClass('listening');
            $('#voiceInputBtn i').removeClass('fa-stop').addClass('fa-microphone');
            $('#voiceInputBtn').attr('title', 'Voice Input');
        };
    }

    // Voice Input Button Click Handler
    $('#voiceInputBtn').click(function() {
        if (!recognition) {
            alert('Sorry, speech recognition is not supported in your browser.');
            return;
        }

        if ($(this).hasClass('listening')) {
            // Stop listening
            recognition.stop();
        } else {
            // Start listening
            recognition.start();
        }
    });
```



```
        // Add Play Audio button for assistant messages
        const playButton = $('<button class="btn btn-sm btn-outline-secondary mt-2 play-audio-btn"><i class="fas fa-volume-up me-1"></i>Play Audio</button>');

        // Add click event to play audio
        playButton.on('click', function() {
            const textToSpeak = content.replace(/```[\s\S]*?```/g, ''); // Remove code blocks

            // Check if SpeechSynthesis is supported
            if ('speechSynthesis' in window) {
                // If button shows "Stop Audio", stop playback and reset
                if ($(this).find('i').hasClass('fa-stop')) {
                    window.speechSynthesis.cancel();
                    $(this).html('<i class="fas fa-volume-up me-1"></i>Play Audio');
                    return false;
                }

                // Create a new speech synthesis utterance
                const utterance = new SpeechSynthesisUtterance(textToSpeak);

                // Optional: Set properties for the speech      You, 2 hours ago • Added text to speech
                utterance.rate = 1.0; // Speed
                utterance.pitch = 1.0; // Pitch

                // Stop any currently playing audio
                window.speechSynthesis.cancel();

                // Play the audio
                window.speechSynthesis.speak(utterance);

                // Change button text while playing
                $(this).html('<i class="fas fa-stop me-1"></i>Stop Audio');
```

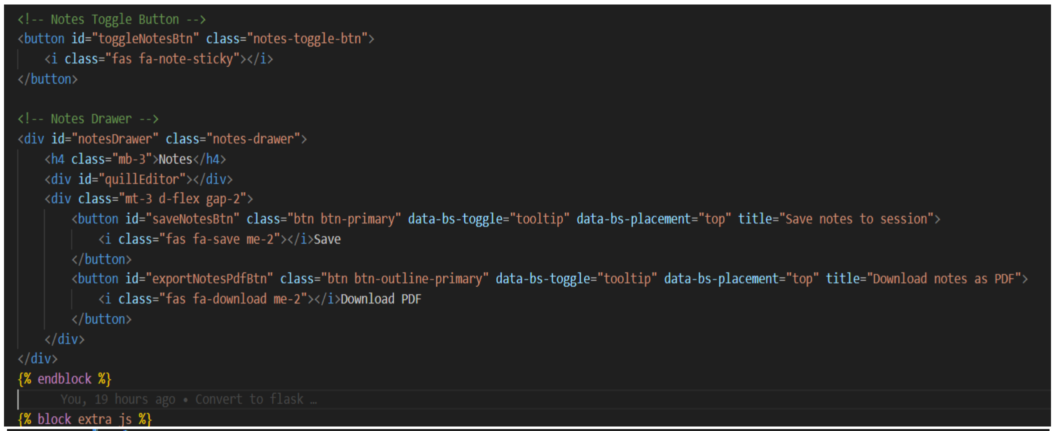


```
            // Change button back when done speaking
            utterance.onend = function() {
                playButton.html('<i class="fas fa-volume-up me-1"></i>Play Audio');
            };
        } else {
            alert('Sorry, your browser does not support text-to-speech!');
        }

        return false;
    });

    messageElement.append(playButton);
```

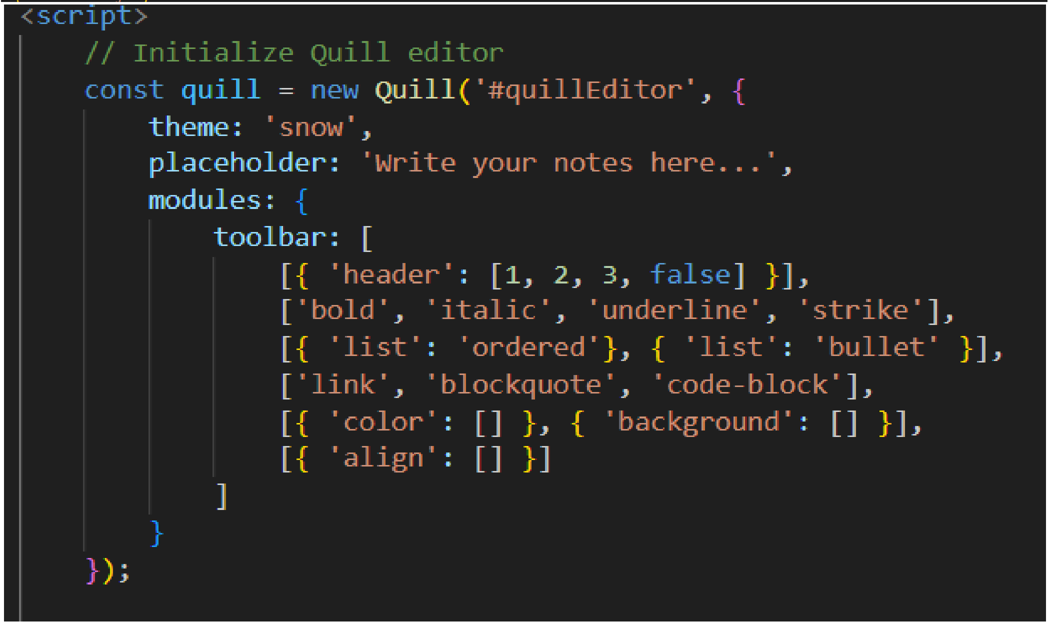
5. Knowledge Toolkit

Components:

- Notes Editor: Provides a simple interface for users to manually type or paste any content (e.g., model responses, summaries). Users can download these notes as pdf files. This promotes convenient personal knowledge organization without switching contexts.

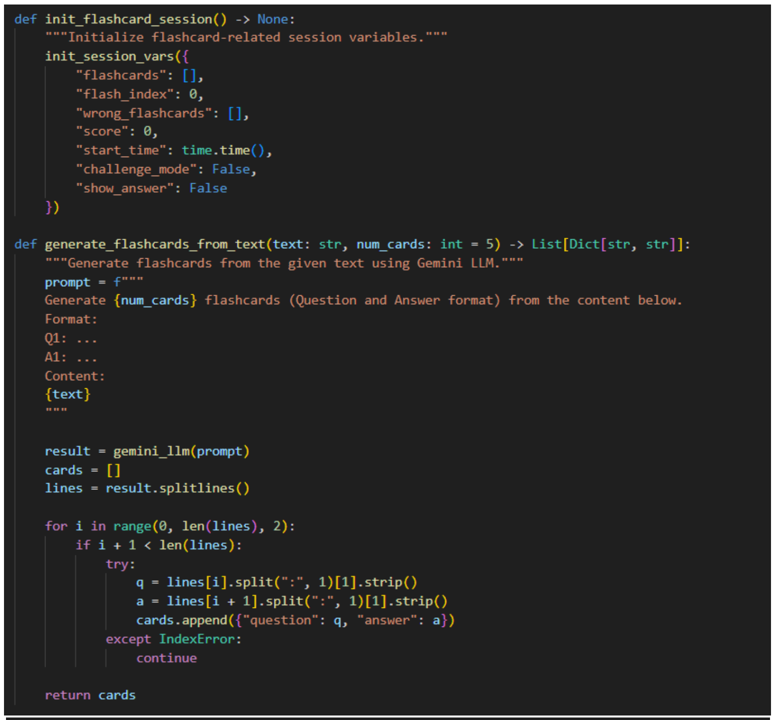


```
<!-- Notes Toggle Button -->
<button id="toggleNotesBtn" class="notes-toggle-btn">
    <i class="fas fa-note-sticky"></i>
</button>

<!-- Notes Drawer -->
<div id="notesDrawer" class="notes-drawer">
    <h4 class="mb-3">Notes</h4>
    <div id="quillEditor"></div>
    <div class="mt-3 d-flex gap-2">
        <button id="saveNotesBtn" class="btn btn-primary" data-bs-toggle="tooltip" data-bs-placement="top" title="Save notes to session">
            <i class="fas fa-save me-2"></i>Save
        </button>
        <button id="exportNotesPdfBtn" class="btn btn-outline-primary" data-bs-toggle="tooltip" data-bs-placement="top" title="Download notes as PDF">
            <i class="fas fa-download me-2"></i>Download PDF
        </button>
    </div>
</div>
{% endblock %}
       You, 19 hours ago • Convert to flask …
{% block extra js %}
```

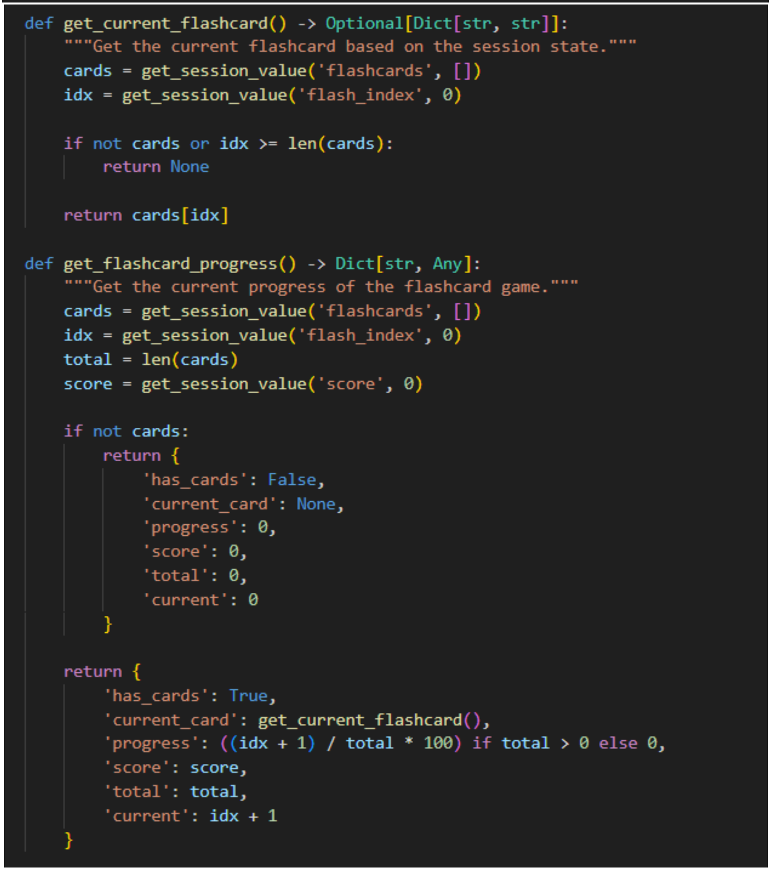


```
<script>
    // Initialize Quill editor
    const quill = new Quill('#quillEditor', {
        theme: 'snow',
        placeholder: 'Write your notes here...',
        modules: {
            toolbar: [
                [{ 'header': [1, 2, 3, false] }],
                ['bold', 'italic', 'underline', 'strike'],
                [{ 'list': 'ordered'}, { 'list': 'bullet' }],
                ['link', 'blockquote', 'code-block'],
                [{ 'color': [] }, { 'background': [] }],
                [{ 'align': [] }]
            ]
        }
    });
```

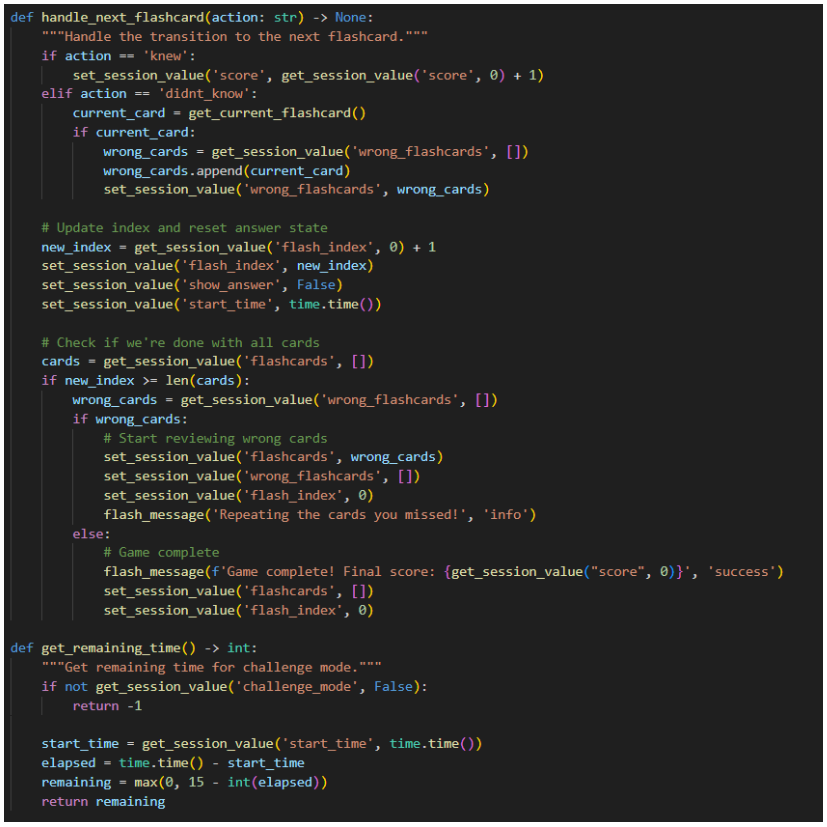
- Flashcard Generator: Converts content into Q&A pairs to support spaced repetition learning.



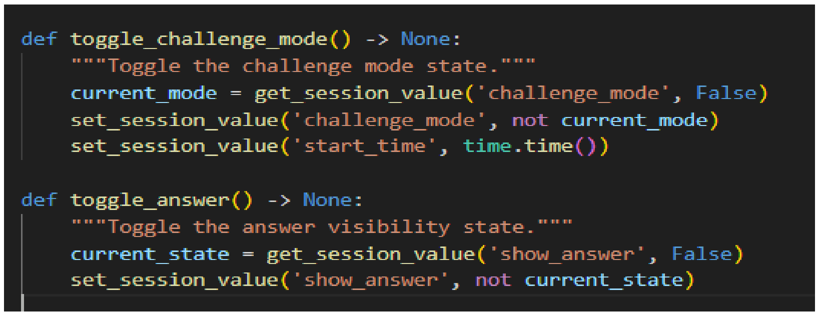
```python
def init_flashcard_session() -> None:
    """Initialize flashcard-related session variables."""
    init_session_vars({
        "flashcards": [],
        "flash_index": 0,
        "wrong_flashcards": [],
        "score": 0,
        "start_time": time.time(),
        "challenge_mode": False,
        "show_answer": False
    })

def generate_flashcards_from_text(text: str, num_cards: int = 5) -> List[Dict[str, str]]:
    """Generate flashcards from the given text using Gemini LLM."""
    prompt = f"""
    Generate {num_cards} flashcards (Question and Answer format) from the content below.
    Format:
    Q1: ...
    A1: ...
    Content:
    {text}
    """

    result = gemini_llm(prompt)
    cards = []
    lines = result.splitlines()

    for i in range(0, len(lines), 2):
        if i + 1 < len(lines):
            try:
                q = lines[i].split(":", 1)[1].strip()
                a = lines[i + 1].split(":", 1)[1].strip()
                cards.append({"question": q, "answer": a})
            except IndexError:
                continue

    return cards
```



```python
def get_current_flashcard() -> Optional[Dict[str, str]]:
    """Get the current flashcard based on the session state."""
    cards = get_session_value('flashcards', [])
    idx = get_session_value('flash_index', 0)

    if not cards or idx >= len(cards):
        return None

    return cards[idx]

def get_flashcard_progress() -> Dict[str, Any]:
    """Get the current progress of the flashcard game."""
    cards = get_session_value('flashcards', [])
    idx = get_session_value('flash_index', 0)
    total = len(cards)
    score = get_session_value('score', 0)

    if not cards:
        return {
            'has_cards': False,
            'current_card': None,
            'progress': 0,
            'score': 0,
            'total': 0,
            'current': 0
        }

    return {
        'has_cards': True,
        'current_card': get_current_flashcard(),
        'progress': ((idx + 1) / total * 100) if total > 0 else 0,
        'score': score,
        'total': total,
        'current': idx + 1
    }
```



```python
def handle_next_flashcard(action: str) -> None:
    """Handle the transition to the next flashcard."""
    if action == 'knew':
        set_session_value('score', get_session_value('score', 0) + 1)
    elif action == 'didnt_know':
        current_card = get_current_flashcard()
        if current_card:
            wrong_cards = get_session_value('wrong_flashcards', [])
            wrong_cards.append(current_card)
            set_session_value('wrong_flashcards', wrong_cards)

    # Update index and reset answer state
    new_index = get_session_value('flash_index', 0) + 1
    set_session_value('flash_index', new_index)
    set_session_value('show_answer', False)
    set_session_value('start_time', time.time())

    # Check if we're done with all cards
    cards = get_session_value('flashcards', [])
    if new_index >= len(cards):
        wrong_cards = get_session_value('wrong_flashcards', [])
        if wrong_cards:
            # Start reviewing wrong cards
            set_session_value('flashcards', wrong_cards)
            set_session_value('wrong_flashcards', [])
            set_session_value('flash_index', 0)
            flash_message('Repeating the cards you missed!', 'info')
        else:
            # Game complete
            flash_message(f'Game complete! Final score: {get_session_value("score", 0)}', 'success')
            set_session_value('flashcards', [])
            set_session_value('flash_index', 0)

def get_remaining_time() -> int:
    """Get remaining time for challenge mode."""
    if not get_session_value('challenge_mode', False):
        return -1

    start_time = get_session_value('start_time', time.time())
    elapsed = time.time() - start_time
    remaining = max(0, 15 - int(elapsed))
    return remaining
```



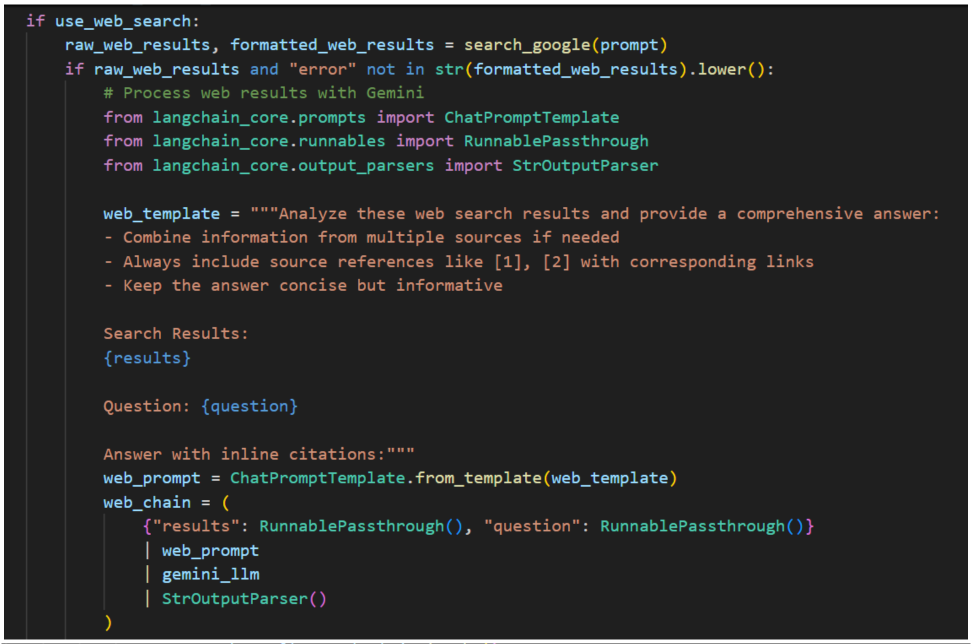
```python
def toggle_challenge_mode() -> None:
    """Toggle the challenge mode state."""
    current_mode = get_session_value('challenge_mode', False)
    set_session_value('challenge_mode', not current_mode)
    set_session_value('start_time', time.time())

def toggle_answer() -> None:
    """Toggle the answer visibility state."""
    current_state = get_session_value('show_answer', False)
    set_session_value('show_answer', not current_state)
```

Justification: Transforms passive information into active learning aids, supporting better retention and efficient study workflows.

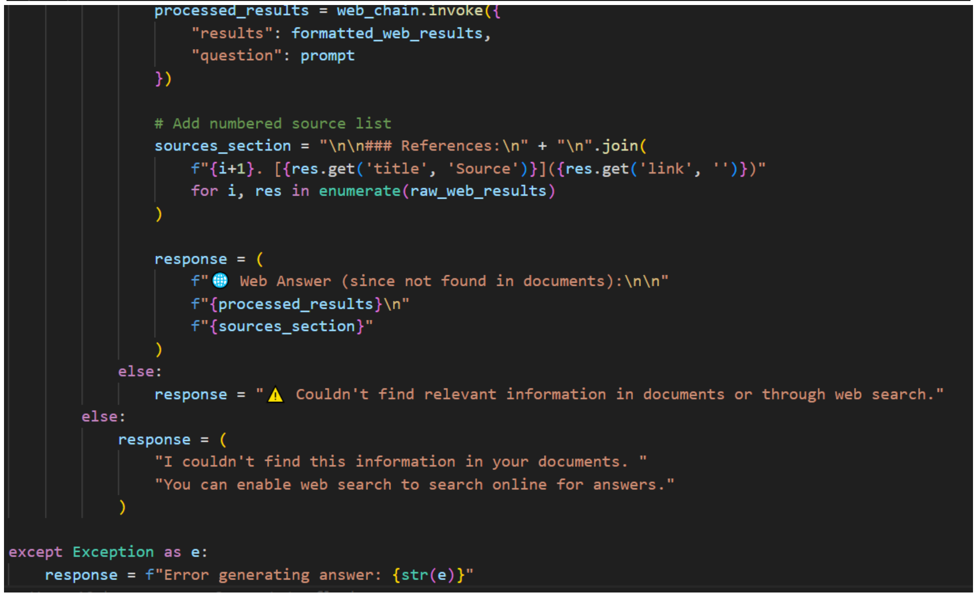
6. Web Scraper

Function: Extracts real-time data from online sources to supplement local document knowledge. Users can either manually input a URL for targeted scraping or enable a "Web Search" toggle. If enabled and no relevant answer is found in the uploaded documents, the system automatically fetches information from the web along with reference links.

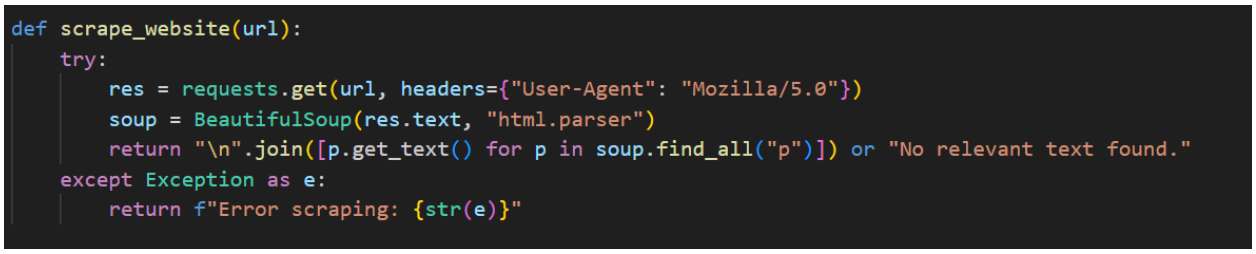
Justification: Ensures up-to-date, comprehensive answers by extending beyond static uploads and integrating live web information when needed.



```python
if use_web_search:
    raw_web_results, formatted_web_results = search_google(prompt)
    if raw_web_results and "error" not in str(formatted_web_results).lower():
        # Process web results with Gemini
        from langchain_core.prompts import ChatPromptTemplate
        from langchain_core.runnables import RunnablePassthrough
        from langchain_core.output_parsers import StrOutputParser

        web_template = """Analyze these web search results and provide a comprehensive answer:
        - Combine information from multiple sources if needed
        - Always include source references like [1], [2] with corresponding links
        - Keep the answer concise but informative

        Search Results:
        {results}

        Question: {question}

        Answer with inline citations:"""
        web_prompt = ChatPromptTemplate.from_template(web_template)
        web_chain = (
            {"results": RunnablePassthrough(), "question": RunnablePassthrough()}
            | web_prompt
            | gemini_llm
            | StrOutputParser()
        )
```



```python
                processed_results = web_chain.invoke({
                    "results": formatted_web_results,
                    "question": prompt
                })

                # Add numbered source list
                sources_section = "\n\n### References:\n" + "\n".join(
                    f"{i+1}. [{res.get('title', 'Source')}]({res.get('link', '')})"
                    for i, res in enumerate(raw_web_results)
                )

                response = (
                    f"🌐 Web Answer (since not found in documents):\n\n"
                    f"{processed_results}\n"
                    f"{sources_section}"
                )
            else:
                response = "⚠️ Couldn't find relevant information in documents or through web search."
        else:
            response = (
                "I couldn't find this information in your documents. "
                "You can enable web search to search online for answers."
            )

except Exception as e:
    response = f"Error generating answer: {str(e)}"
```

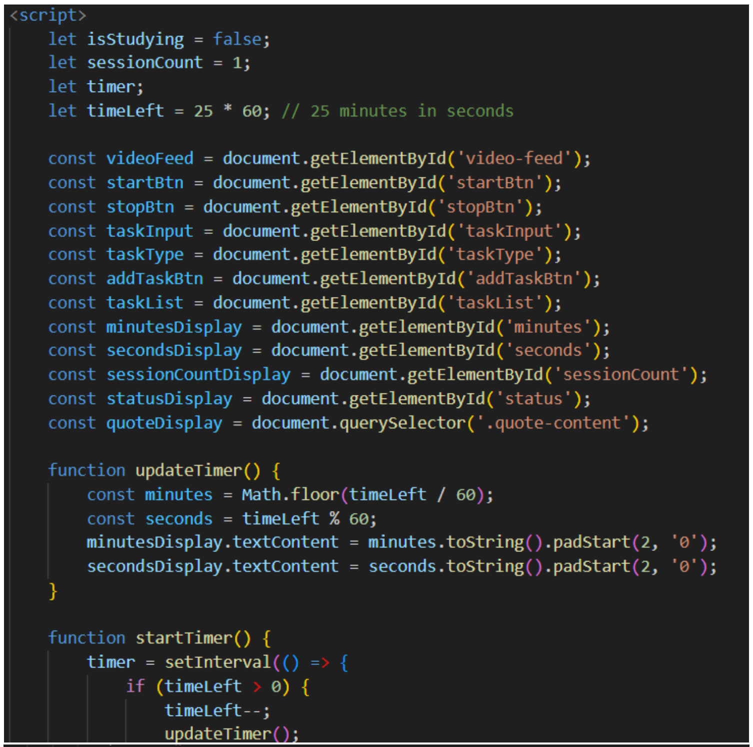


```python
def scrape_website(url):
    try:
        res = requests.get(url, headers={"User-Agent": "Mozilla/5.0"})
        soup = BeautifulSoup(res.text, "html.parser")
        return "\n".join([p.get_text() for p in soup.find_all("p")]) or "No relevant text found."
    except Exception as e:
        return f"Error scraping: {str(e)}"
```

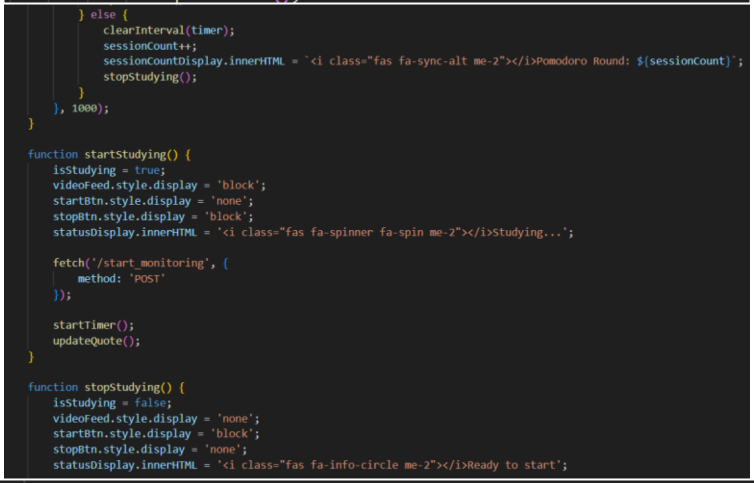
7. Pomodoro-Based Task Manager

Function: Implements 25-minute focused work sessions followed by 5-minute breaks. Tracks task history, completion analytics, and measures task difficulty based on user fatigue indicators such as yawning and eye closure frequency during sessions.

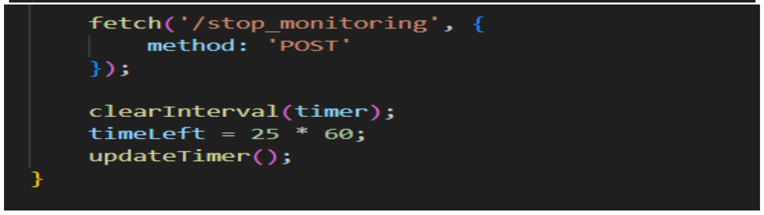
Justification: Encourages time management and cognitive stamina by linking productivity with physical wellness signals.



```
<script>
    let isStudying = false;
    let sessionCount = 1;
    let timer;
    let timeLeft = 25 * 60; // 25 minutes in seconds

    const videoFeed = document.getElementById('video-feed');
    const startBtn = document.getElementById('startBtn');
    const stopBtn = document.getElementById('stopBtn');
    const taskInput = document.getElementById('taskInput');
    const taskType = document.getElementById('taskType');
    const addTaskBtn = document.getElementById('addTaskBtn');
    const taskList = document.getElementById('taskList');
    const minutesDisplay = document.getElementById('minutes');
    const secondsDisplay = document.getElementById('seconds');
    const sessionCountDisplay = document.getElementById('sessionCount');
    const statusDisplay = document.getElementById('status');
    const quoteDisplay = document.querySelector('.quote-content');

    function updateTimer() {
        const minutes = Math.floor(timeLeft / 60);
        const seconds = timeLeft % 60;
        minutesDisplay.textContent = minutes.toString().padStart(2, '0');
        secondsDisplay.textContent = seconds.toString().padStart(2, '0');
    }

    function startTimer() {
        timer = setInterval(() => {
            if (timeLeft > 0) {
                timeLeft--;
                updateTimer();
```



```
            } else {
                clearInterval(timer);
                sessionCount++;
                sessionCountDisplay.innerHTML = `<i class="fas fa-sync-alt me-2"></i>Pomodoro Round: ${sessionCount}`;
                stopStudying();
            }
        }, 1000);
    }

    function startStudying() {
        isStudying = true;
        videoFeed.style.display = 'block';
        startBtn.style.display = 'none';
        stopBtn.style.display = 'block';
        statusDisplay.innerHTML = '<i class="fas fa-spinner fa-spin me-2"></i>Studying...';

        fetch('/start_monitoring', {
            method: 'POST'
        });

        startTimer();
        updateQuote();
    }

    function stopStudying() {
        isStudying = false;
        videoFeed.style.display = 'none';
        startBtn.style.display = 'block';
        stopBtn.style.display = 'none';
        statusDisplay.innerHTML = '<i class="fas fa-info-circle me-2"></i>Ready to start';
```

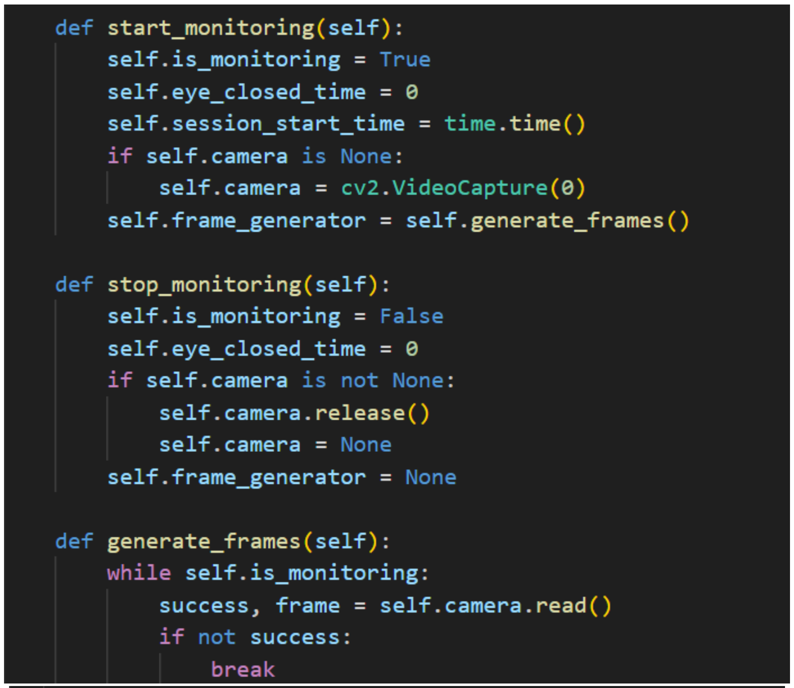


```
        fetch('/stop_monitoring', {
            method: 'POST'
        });

        clearInterval(timer);
        timeLeft = 25 * 60;
        updateTimer();
    }
```

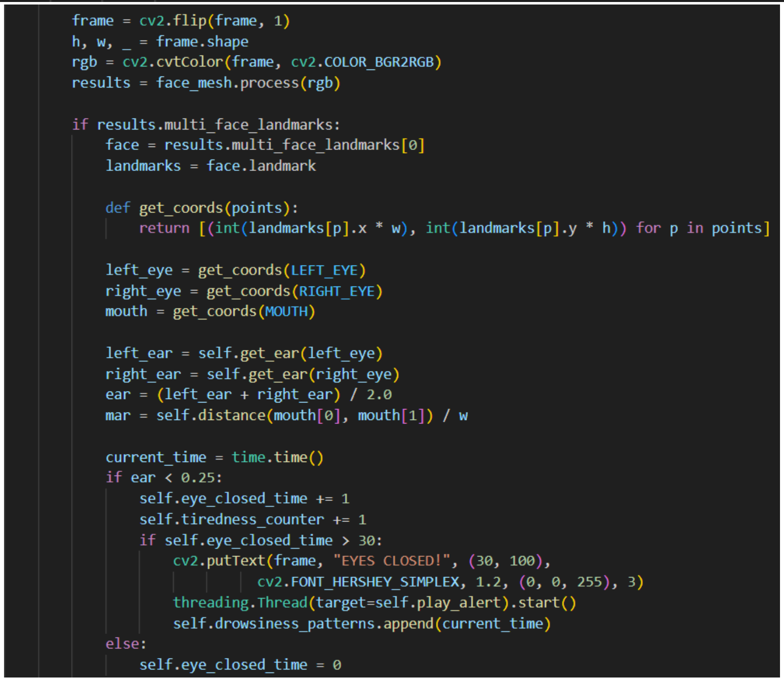
8. Fatigue Detection System (Eye Closure & Yawning Detection)

Function: Uses webcam-based computer vision to detect drowsiness signs like prolonged eye closure or frequent yawning. Triggers audio alerts and motivational messages personalized according to the detected fatigue level and current task type.

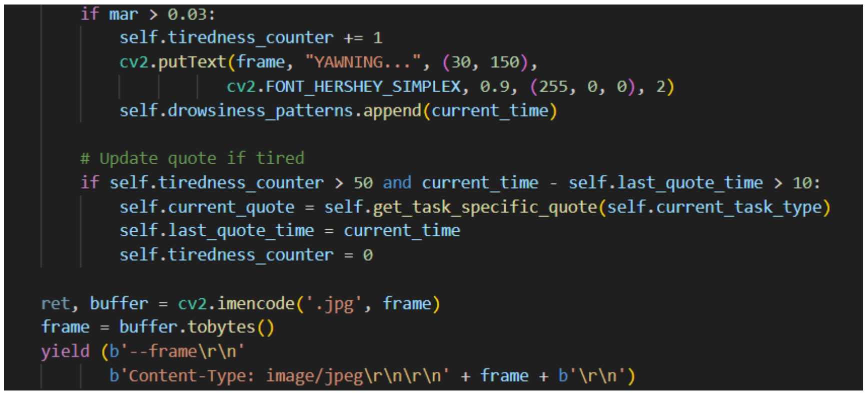
Justification: Supports user wellness by preventing burnout and promoting timely breaks. Fatigue-based prompts adapt to the user's task context (e.g., reading or coding), enhancing sustained attention and productivity.



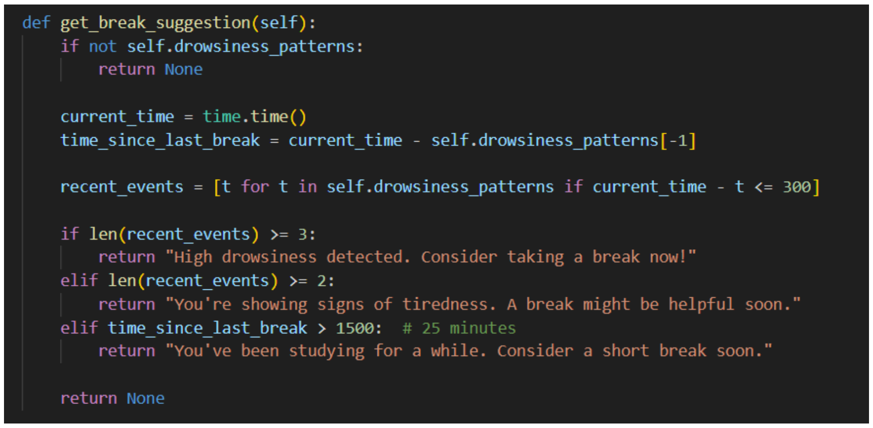
```
def start_monitoring(self):
    self.is_monitoring = True
    self.eye_closed_time = 0
    self.session_start_time = time.time()
    if self.camera is None:
        self.camera = cv2.VideoCapture(0)
    self.frame_generator = self.generate_frames()

def stop_monitoring(self):
    self.is_monitoring = False
    self.eye_closed_time = 0
    if self.camera is not None:
        self.camera.release()
        self.camera = None
    self.frame_generator = None

def generate_frames(self):
    while self.is_monitoring:
        success, frame = self.camera.read()
        if not success:
            break
```



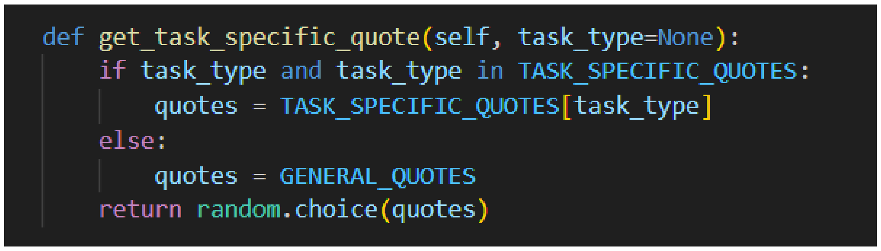
```
        frame = cv2.flip(frame, 1)
        h, w, _ = frame.shape
        rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        results = face_mesh.process(rgb)

        if results.multi_face_landmarks:
            face = results.multi_face_landmarks[0]
            landmarks = face.landmark

            def get_coords(points):
                return [(int(landmarks[p].x * w), int(landmarks[p].y * h)) for p in points]

            left_eye = get_coords(LEFT_EYE)
            right_eye = get_coords(RIGHT_EYE)
            mouth = get_coords(MOUTH)

            left_ear = self.get_ear(left_eye)
            right_ear = self.get_ear(right_eye)
            ear = (left_ear + right_ear) / 2.0
            mar = self.distance(mouth[0], mouth[1]) / w

            current_time = time.time()
            if ear < 0.25:
                self.eye_closed_time += 1
                self.tiredness_counter += 1
                if self.eye_closed_time > 30:
                    cv2.putText(frame, "EYES CLOSED!", (30, 100),
                              cv2.FONT_HERSHEY_SIMPLEX, 1.2, (0, 0, 255), 3)
                    threading.Thread(target=self.play_alert).start()
                    self.drowsiness_patterns.append(current_time)
            else:
                self.eye_closed_time = 0
```



```
        if mar > 0.03:
            self.tiredness_counter += 1
            cv2.putText(frame, "YAWNING...", (30, 150),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)
            self.drowsiness_patterns.append(current_time)

        # Update quote if tired
        if self.tiredness_counter > 50 and current_time - self.last_quote_time > 10:
            self.current_quote = self.get_task_specific_quote(self.current_task_type)
            self.last_quote_time = current_time
            self.tiredness_counter = 0

    ret, buffer = cv2.imencode('.jpg', frame)
    frame = buffer.tobytes()
    yield (b'--frame\r\n'
           b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
```



```
def get_break_suggestion(self):
    if not self.drowsiness_patterns:
        return None

    current_time = time.time()
    time_since_last_break = current_time - self.drowsiness_patterns[-1]

    recent_events = [t for t in self.drowsiness_patterns if current_time - t <= 300]

    if len(recent_events) >= 3:
        return "High drowsiness detected. Consider taking a break now!"
    elif len(recent_events) >= 2:
        return "You're showing signs of tiredness. A break might be helpful soon."
    elif time_since_last_break > 1500:  # 25 minutes
        return "You've been studying for a while. Consider a short break soon."

    return None
```

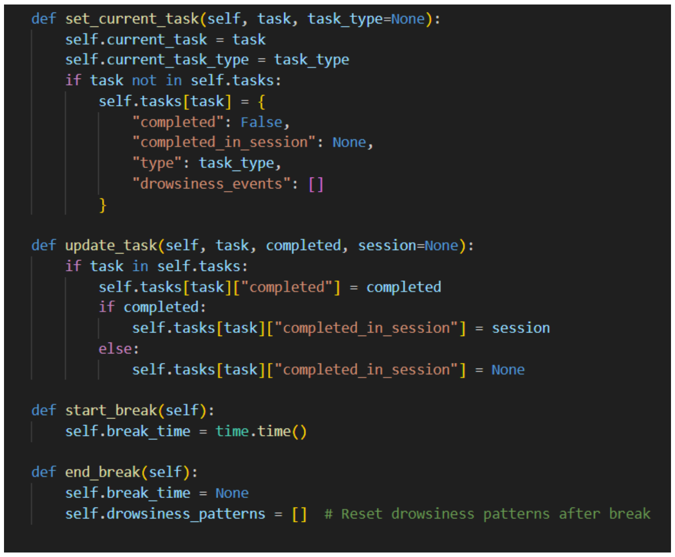


```
def get_task_specific_quote(self, task_type=None):
    if task_type and task_type in TASK_SPECIFIC_QUOTES:
        quotes = TASK_SPECIFIC_QUOTES[task_type]
    else:
        quotes = GENERAL_QUOTES
    return random.choice(quotes)
```

9. Task Management System

Function: Enables users to add, update, complete, and prioritize tasks with detailed status and fatigue tracking. Records how much fatigue each task induces, helping users understand which tasks are more taxing.

Justification: Helps users organize workflow more effectively by aligning task difficulty and fatigue levels with focus sessions. This empowers smarter scheduling and rest breaks tailored to individual task demands.

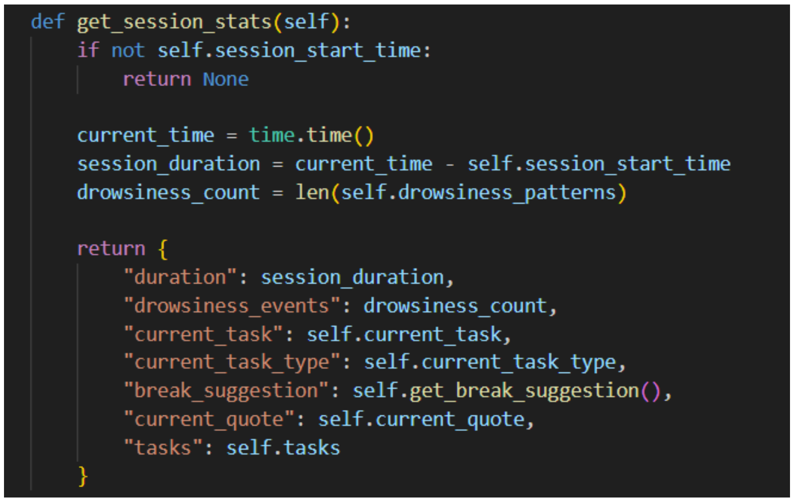


```
def set_current_task(self, task, task_type=None):
    self.current_task = task
    self.current_task_type = task_type
    if task not in self.tasks:
        self.tasks[task] = {
            "completed": False,
            "completed_in_session": None,
            "type": task_type,
            "drowsiness_events": []
        }

def update_task(self, task, completed, session=None):
    if task in self.tasks:
        self.tasks[task]["completed"] = completed
        if completed:
            self.tasks[task]["completed_in_session"] = session
        else:
            self.tasks[task]["completed_in_session"] = None

def start_break(self):
    self.break_time = time.time()

def end_break(self):
    self.break_time = None
    self.drowsiness_patterns = []  # Reset drowsiness patterns after break
```

10. Productivity Insights

Function: Monitors work sessions, breaks, task progress, and fatigue levels. Displays motivational quotes that are specifically tailored to the current task and user's fatigue state—for example, encouraging breaks if frequent yawns are detected during a session, or task-specific motivation (e.g., focused encouragement for reading tasks).

Justification: Increases motivation and self-awareness by providing personalized feedback that reflects both productivity and wellness. This holistic approach supports sustainable progress and improved habit formation.

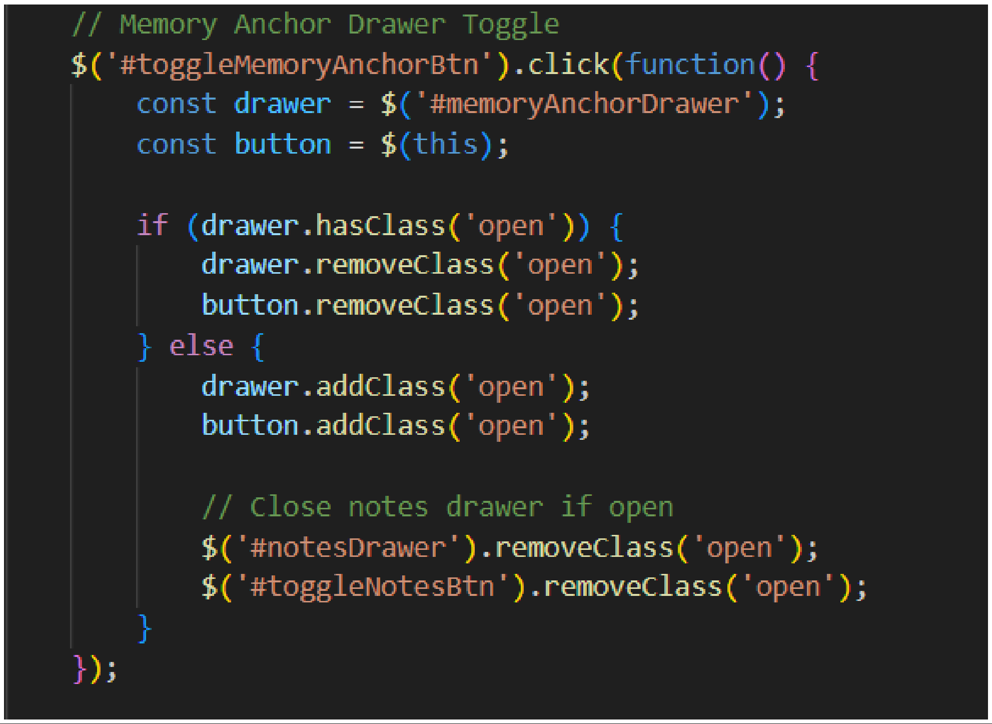


```
def get_session_stats(self):
    if not self.session_start_time:
        return None

    current_time = time.time()
    session_duration = current_time - self.session_start_time
    drowsiness_count = len(self.drowsiness_patterns)

    return {
        "duration": session_duration,
        "drowsiness_events": drowsiness_count,
        "current_task": self.current_task,
        "current_task_type": self.current_task_type,
        "break_suggestion": self.get_break_suggestion(),
        "current_quote": self.current_quote,
        "tasks": self.tasks
    }
```

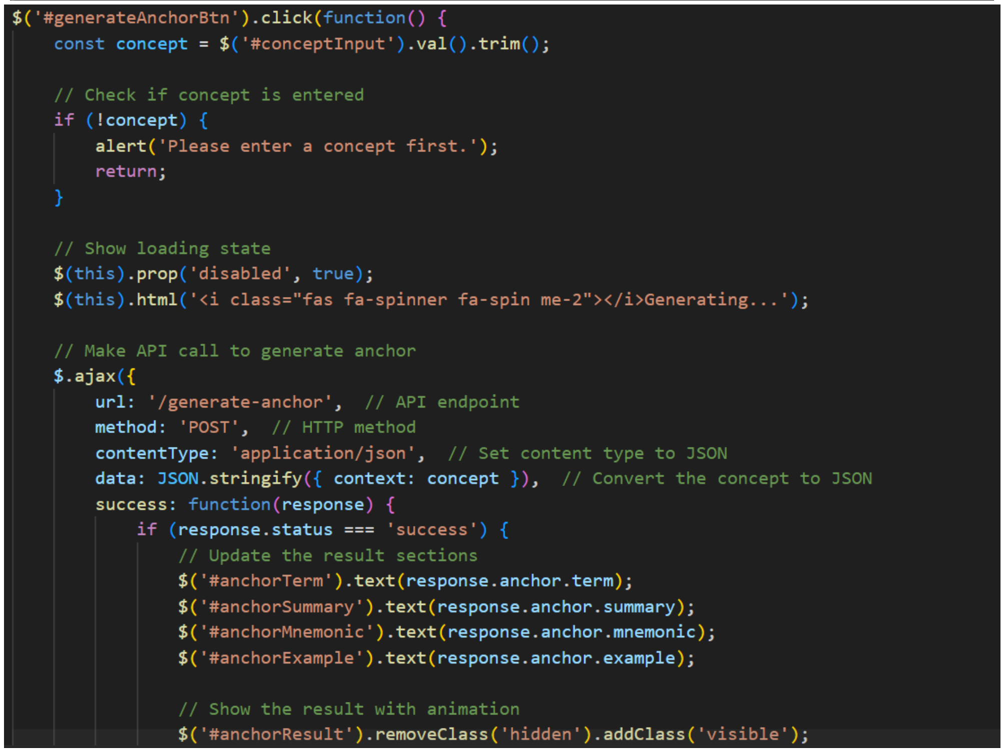
11. Memory Anchor Generator

Function: Generates a uniquely named "anchor" for any concept, along with a brief summary, mnemonic, and relatable example to improve memory retention. Enhances revision, learning, and recall for complex topics.
Justification: Aids users in consolidating key ideas through cognitive techniques. Especially helpful

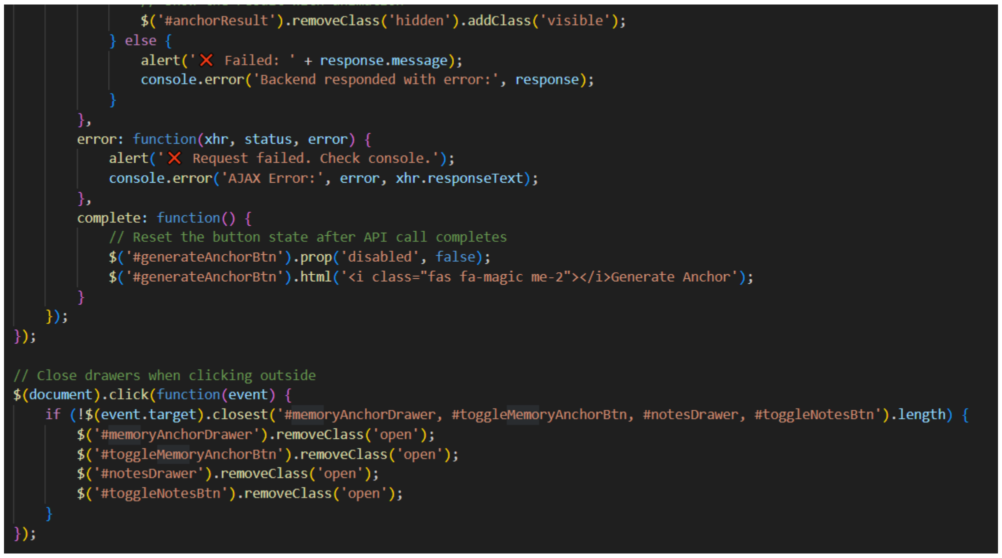
for learners dealing with technical jargon or abstract system components, enabling long-term understanding and active recall.



```
// Memory Anchor Drawer Toggle
$('#toggleMemoryAnchorBtn').click(function() {
    const drawer = $('#memoryAnchorDrawer');
    const button = $(this);

    if (drawer.hasClass('open')) {
        drawer.removeClass('open');
        button.removeClass('open');
    } else {
        drawer.addClass('open');
        button.addClass('open');

        // Close notes drawer if open
        $('#notesDrawer').removeClass('open');
        $('#toggleNotesBtn').removeClass('open');
    }
});
```



```
$('#generateAnchorBtn').click(function() {
    const concept = $('#conceptInput').val().trim();

    // Check if concept is entered
    if (!concept) {
        alert('Please enter a concept first.');
        return;
    }

    // Show loading state
    $(this).prop('disabled', true);
    $(this).html('<i class="fas fa-spinner fa-spin me-2"></i>Generating...');

    // Make API call to generate anchor
    $.ajax({
        url: '/generate-anchor',  // API endpoint
        method: 'POST',  // HTTP method
        contentType: 'application/json',  // Set content type to JSON
        data: JSON.stringify({ context: concept }),  // Convert the concept to JSON
        success: function(response) {
            if (response.status === 'success') {
                // Update the result sections
                $('#anchorTerm').text(response.anchor.term);
                $('#anchorSummary').text(response.anchor.summary);
                $('#anchorMnemonic').text(response.anchor.mnemonic);
                $('#anchorExample').text(response.anchor.example);

                // Show the result with animation
                $('#anchorResult').removeClass('hidden').addClass('visible');
```

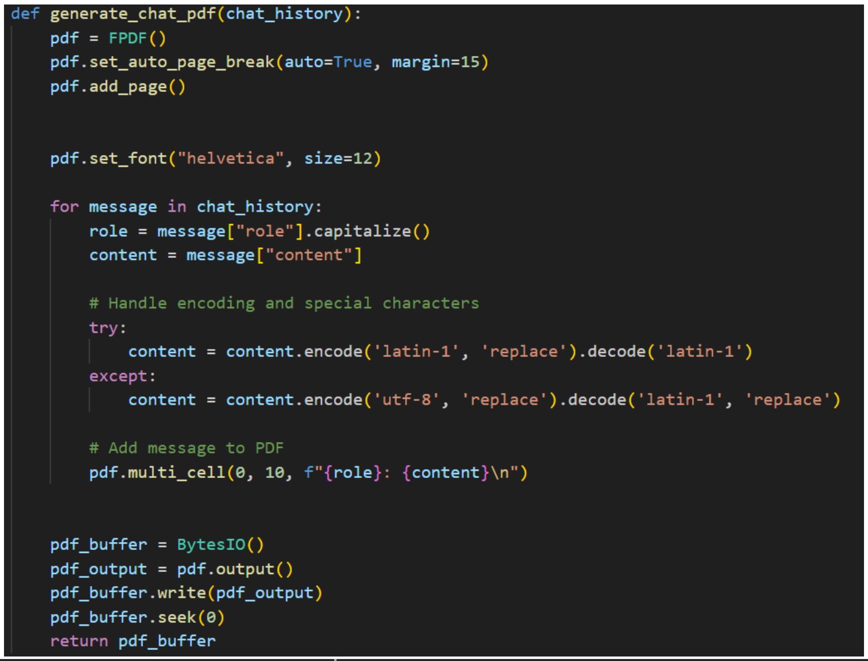


```
                $('#anchorResult').removeClass('hidden').addClass('visible');
            } else {
                alert('❌ Failed: ' + response.message);
                console.error('Backend responded with error:', response);
            }
        },
        error: function(xhr, status, error) {
            alert('❌ Request failed. Check console.');
            console.error('AJAX Error:', error, xhr.responseText);
        },
        complete: function() {
            // Reset the button state after API call completes
            $('#generateAnchorBtn').prop('disabled', false);
            $('#generateAnchorBtn').html('<i class="fas fa-magic me-2"></i>Generate Anchor');
        }
    });
});

// Close drawers when clicking outside
$(document).click(function(event) {
    if (!$(event.target).closest('#memoryAnchorDrawer, #toggleMemoryAnchorBtn, #notesDrawer, #toggleNotesBtn').length) {
        $('#memoryAnchorDrawer').removeClass('open');
        $('#toggleMemoryAnchorBtn').removeClass('open');
        $('#notesDrawer').removeClass('open');
        $('#toggleNotesBtn').removeClass('open');
    }
});
```

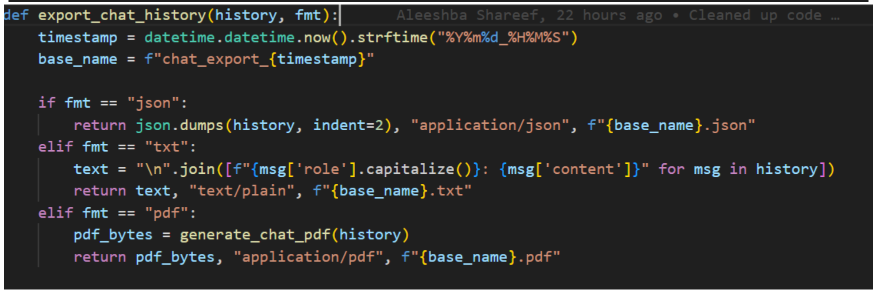
12. Chat Export Module

Function: Allows users to export conversation history in a structured format.

Justification: Helps users archive insights, revisit important queries, and avoid repetition.



```
def generate_chat_pdf(chat_history):
    pdf = FPDF()
    pdf.set_auto_page_break(auto=True, margin=15)
    pdf.add_page()


    pdf.set_font("helvetica", size=12)

    for message in chat_history:
        role = message["role"].capitalize()
        content = message["content"]

        # Handle encoding and special characters
        try:
            content = content.encode('latin-1', 'replace').decode('latin-1')
        except:
            content = content.encode('utf-8', 'replace').decode('latin-1', 'replace')

        # Add message to PDF
        pdf.multi_cell(0, 10, f"{role}: {content}\n")


    pdf_buffer = BytesIO()
    pdf_output = pdf.output()
    pdf_buffer.write(pdf_output)
    pdf_buffer.seek(0)
    return pdf_buffer
```



```
def export_chat_history(history, fmt):        Aleeshba Shareef, 22 hours ago • Cleaned up code …
    timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
    base_name = f"chat_export_{timestamp}"

    if fmt == "json":
        return json.dumps(history, indent=2), "application/json", f"{base_name}.json"
    elif fmt == "txt":
        text = "\n".join([f"{msg['role'].capitalize()}: {msg['content']}" for msg in history])
        return text, "text/plain", f"{base_name}.txt"
    elif fmt == "pdf":
        pdf_bytes = generate_chat_pdf(history)
        return pdf_bytes, "application/pdf", f"{base_name}.pdf"
```

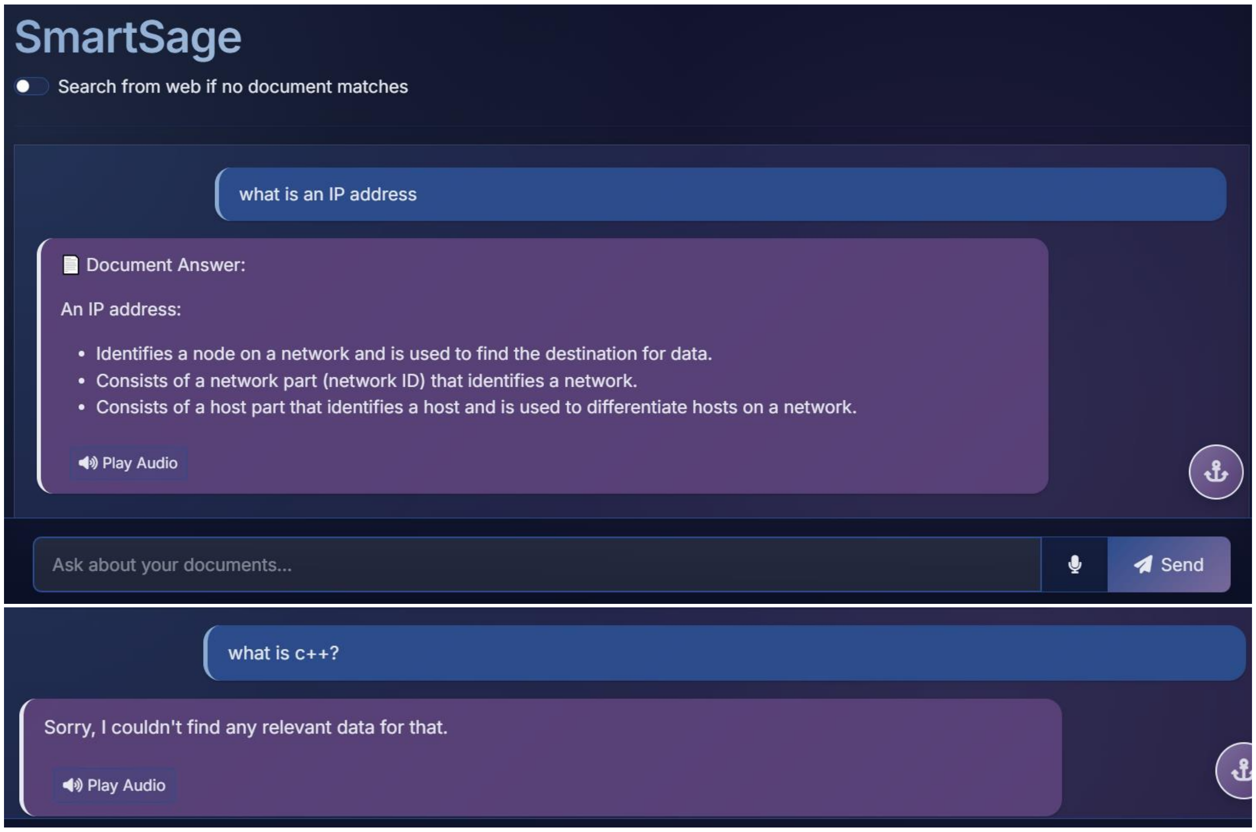
6. AI & Technology Stack Overview

Technology / Concept	Domain / Category	Role in the System
Retrieval-Augmented Generation (RAG)	Natural Language Processing (NLP)	Combines document retrieval with generative models for context-rich, accurate responses.
Multi-query Generation	NLP / Information Retrieval	Reformulates queries into variations to improve recall and answer accuracy.
Query Decomposition	NLP / Prompt Engineering	Splits complex prompts into sub-questions for deeper, structured understanding.
RAG Fusion	NLP / Response Optimization	Merges multiple retrieved answers into a unified, accurate final response.
Speech-to-Text	Accessibility	Converts voice input to text enabling hands-free interaction.
Text-to-Speech	Accessibility / NLP	Converts text responses to audio for enhanced accessibility and learning on the go.
Flashcard Generator	AI in Education / NLP	Generates Q&A flashcards from content to aid active recall and memorization.
Notes Editor	Utility Tool	Enables users to take notes, copy AI responses, and export content easily.
Web Scraper	Information Retrieval	Extracts real-time web data to supplement document-based knowledge. Users can either manually input a URL for targeted scraping or enable a "Web Search" toggle.
Fatigue Detection System	Computer Vision / Wellness	Detects drowsiness signs (eye closure, yawning) via webcam, triggering alerts to maintain focus.
Memory Anchor Generator	Cognitive Learning Tool	Creates anchors, mnemonics, summaries, and examples to support memory and understanding.
Pomodoro-Based Task Manager	Productivity / Time Management	Manages focused work sessions with breaks, tracking task progress and fatigue indicators.

Technology / Concept	**Domain / Category**	**Role in the System**
Task Management System	Workflow Management	CRUD for tasks with prioritization and fatigue tracking per task to monitor difficulty and effort.
Productivity Insights	Behavioral Analytics / Motivation	Tracks sessions, breaks, fatigue, and task progress; delivers tailored motivational quotes.
Chat Export	Utility Tool	Enables exporting chat sessions in multiple formats (txt, pdf, json) for documentation or review.

7. User Interface

The user interface was developed using HTML, CSS, and JavaScript, integrated with Flask for backend communication and dynamic content rendering.


SmartSage
Search from web if no document matches

what is an IP address

Document Answer:

An IP address:

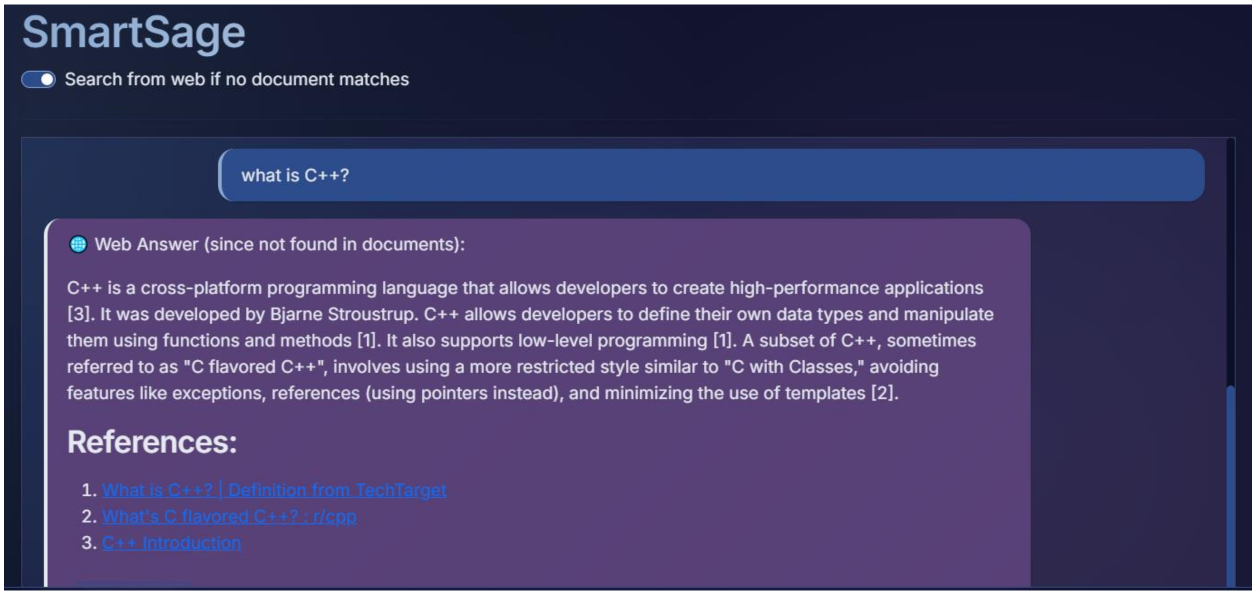
- Identifies a node on a network and is used to find the destination for data.
- Consists of a network part (network ID) that identifies a network.
- Consists of a host part that identifies a host and is used to differentiate hosts on a network.

Play Audio
Ask about your documents...
Send

what is c++?

Sorry, I couldn't find any relevant data for that.

FIGURE 1: CHAT WITH USER UPLOADED DOCUMENT


SmartSage

Search from web if no document matches

what is C++?

Web Answer (since not found in documents):

C++ is a cross-platform programming language that allows developers to create high-performance applications [3]. It was developed by Bjarne Stroustrup. C++ allows developers to define their own data types and manipulate them using functions and methods [1]. It also supports low-level programming [1]. A subset of C++, sometimes referred to as "C flavored C++", involves using a more restricted style similar to "C with Classes," avoiding features like exceptions, references (using pointers instead), and minimizing the use of templates [2].

References:

1. What is C++? | Definition from TechTarget
2. What's C flavored C++? : r/cpp
3. C++ Introduction

FIGURE 2:CHAT ENABLING WEB SEARCH

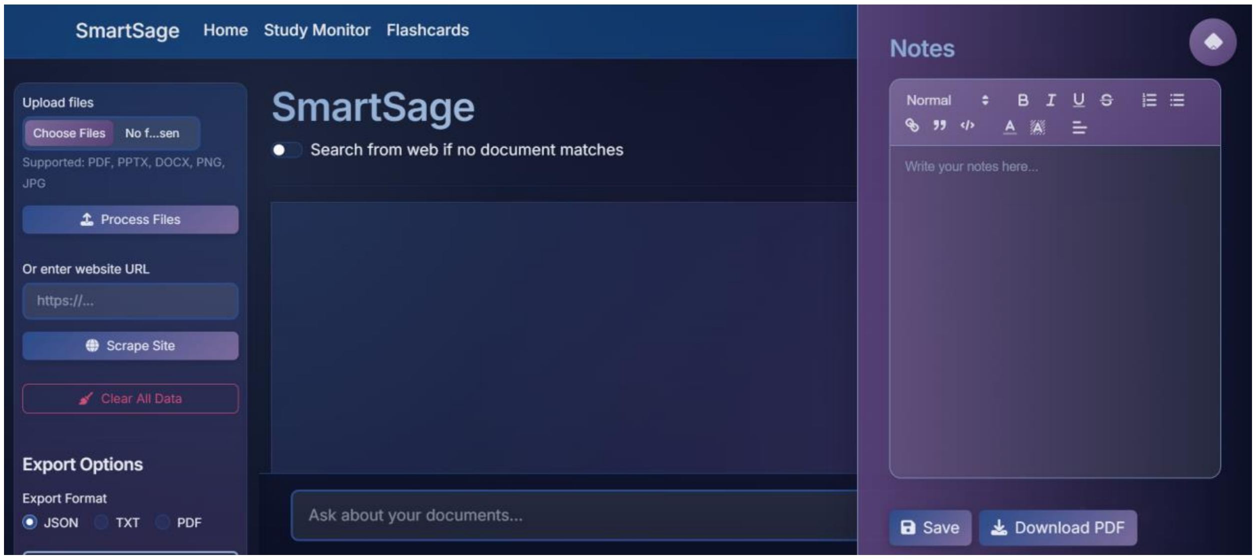
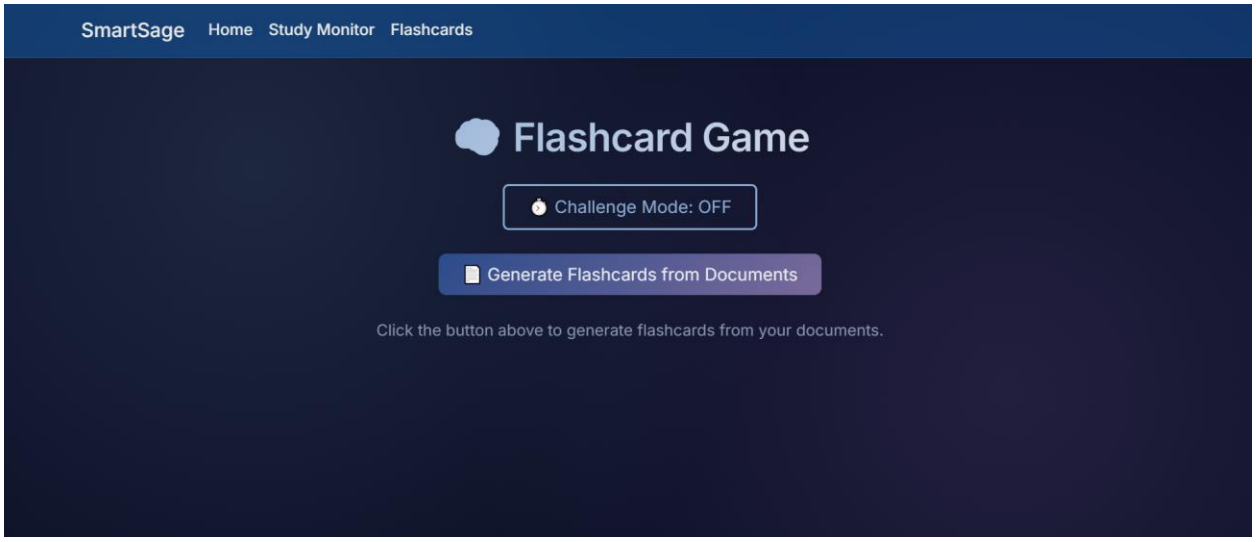
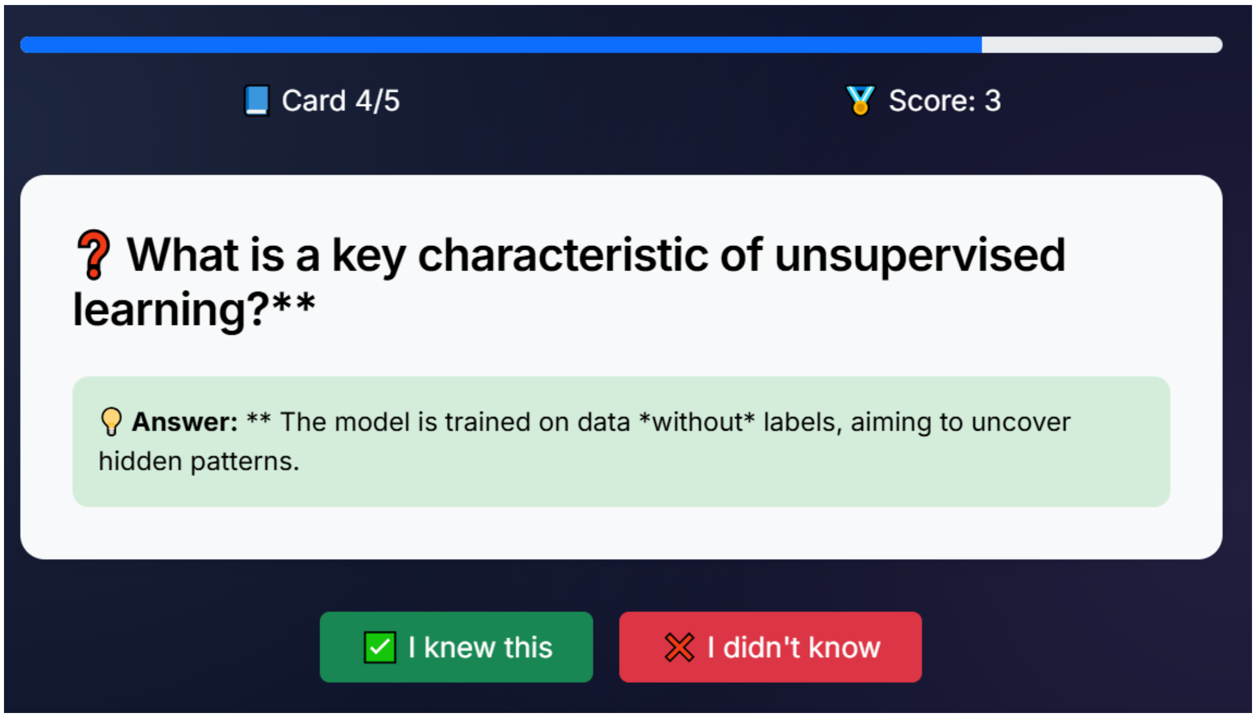


FIGURE 3:NOTES



Click the button above to generate flashcards from your documents.

FIGURE 4: FLASHCARD GENERATION



What is a key characteristic of unsupervised learning?**

Answer: ** The model is trained on data *without* labels, aiming to uncover hidden patterns.

FIGURE 5: FLASHCARD FROM USER UPLOADED DOCUMENT

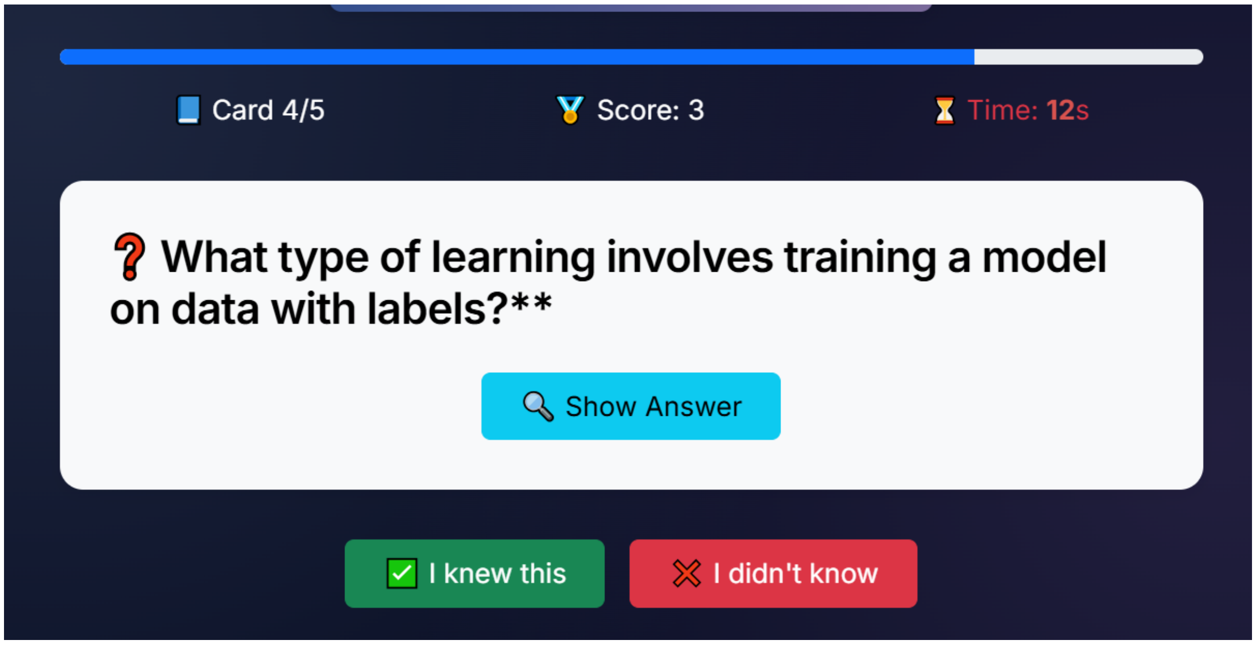

Card 4/5
Score: 3
Time: 12s
❓ What type of learning involves training a model on data with labels?**
Show Answer
I knew this
I didn't know

FIGURE 6: FLASHCARD WITH CHALLENGE MODE

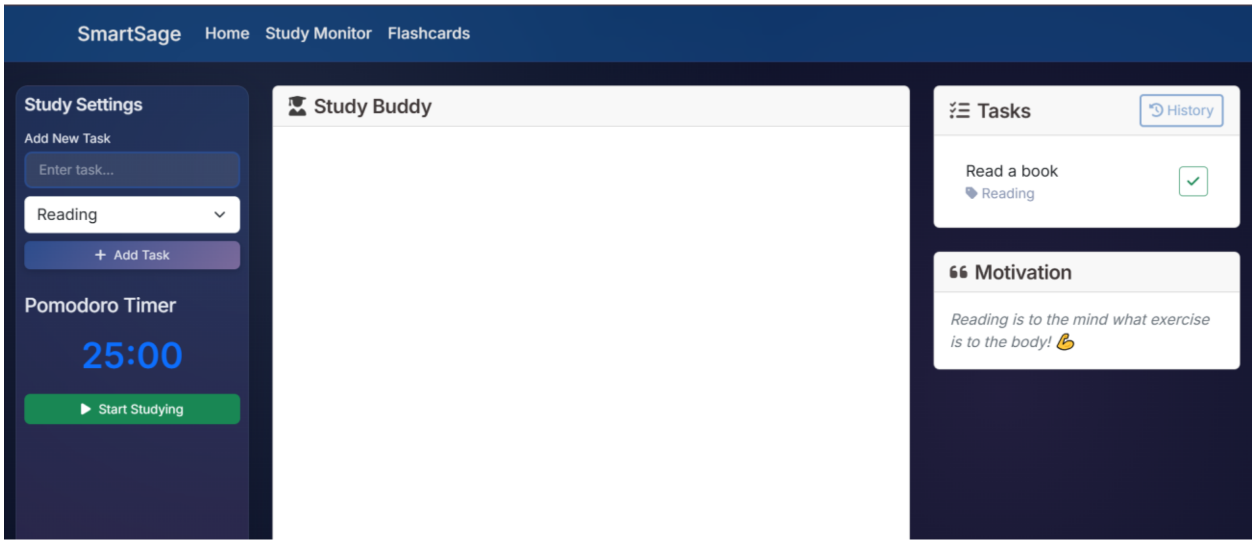
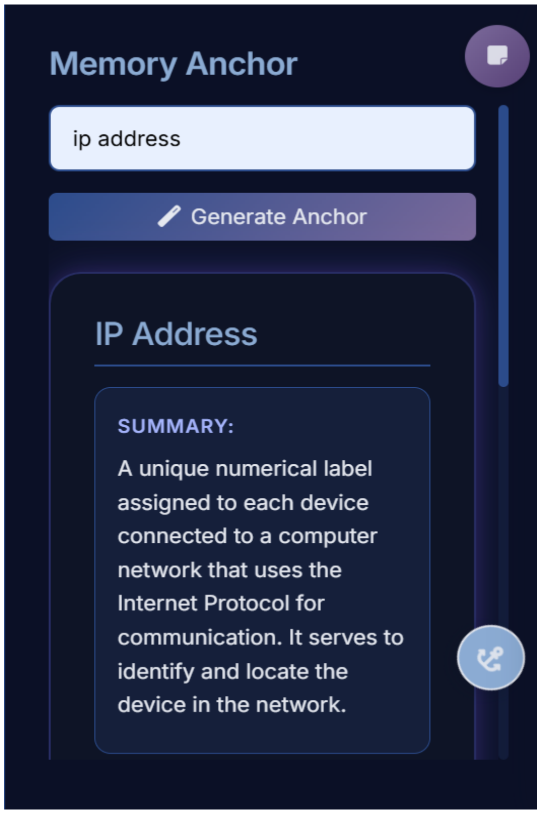


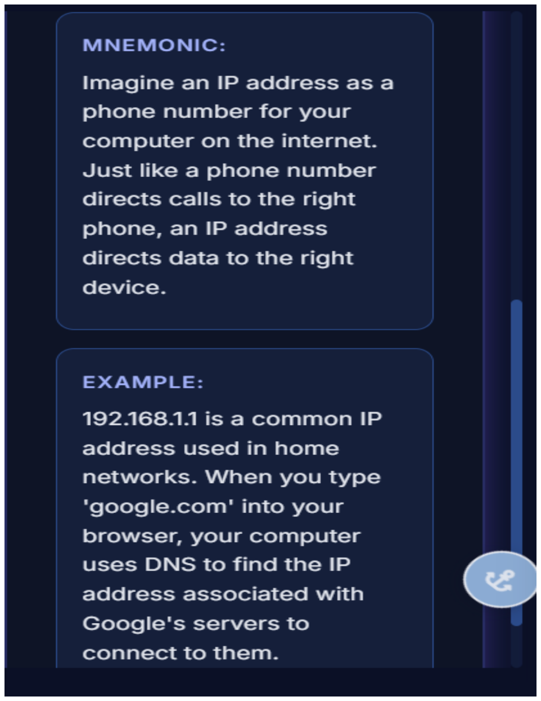
FIGURE 7: STUDY MONITOR



IP Address

SUMMARY:

A unique numerical label assigned to each device connected to a computer network that uses the Internet Protocol for communication. It serves to identify and locate the device in the network.


MNEMONIC:

Imagine an IP address as a phone number for your computer on the internet. Just like a phone number directs calls to the right phone, an IP address directs data to the right device.

EXAMPLE:

192.168.1.1 is a common IP address used in home networks. When you type 'google.com' into your browser, your computer uses DNS to find the IP address associated with Google's servers to connect to them.

FIGURE 8: MEMORY ANCHOR

8. Limitations

Despite offering a robust and intelligent assistant, the system has a few practical and technical constraints:

- **Free Tier API Restrictions:**
  The use of free-tier versions of **Gemini 2.0 Flash** and **Pinecone** limits the number of requests, context length, and retrieval performance. Under high-load conditions, rate limits may affect real-time responsiveness.
- **Basic Fatigue Detection Model:**
  Eye closure and yawning detection is based on simple facial landmarks and heuristics, which may not always be accurate under poor lighting or with certain camera angles.
- **Web Scraping Reliability:**
  Web scraping may be hindered by website structures, CAPTCHA protections, or changing HTML layouts, occasionally affecting content retrieval quality.
- **Limited Session Persistence:**
  Currently, chat context and state are not deeply persistent across sessions due to the use of lightweight storage (in-memory or local), which may hinder long-term progress tracking.

9. Future Enhancements

These enhancements can significantly improve the functionality, scalability, and real-world usefulness of the system:

- **Personalized Learning Paths:**
  Based on flashcard usage, note summaries, and past interactions, the system can suggest custom learning or revision plans for users.
- **Integrated Task Analytics Dashboard:**
  Visual dashboard showing Pomodoro productivity statistics, task completion rates, and user fatigue patterns to enhance productivity tracking.
- **Mobile App Extension:**
  Deploy a cross-platform mobile version for better accessibility and camera-based fatigue detection on the go.
- **Collaborative Mode:**
  Enable group chat sessions or document collaboration so multiple users (e.g., students, teams) can interact with the assistant together.
- **Browser Extension for Web Clipping:**
  Allow users to clip useful content while browsing and send it to the assistant for summarization or flashcard generation directly.

10. Conclusion

The project successfully addresses the need for an intelligent, accessible, and educational AI assistant by blending advanced retrieval techniques with generation models. By integrating voice features, summarization tools, and dynamic web content fetching, it becomes more than just a chatbot—it evolves into a comprehensive knowledge and learning companion. The use of AI principles like RAG, decomposition, and fusion ensures accurate responses, while the educational toolkit empowers users to learn and retain efficiently. The modular architecture ensures that this system can be extended, upgraded, or specialized for various domains such as healthcare, education, or enterprise knowledge management.