Bismillah ar Rahman ar Rahim

Implementation Approaches for Paragraph Headings in Article Generation

Current System Overview

The article generation system currently operates with a hierarchical structure:

1. **Article Level**: The entire article with a main title
2. **Section Level**: Multiple sections, each with its own heading
3. **Paragraph Level**: Multiple paragraphs per section, defined by the `paragraphs_per_section` parameter

Key parameters that influence this structure include:

- `sizesections`: The number of main sections in the article
- `sizeheadings` (or `subsections`): The number of subheadings/points in the outline
- `paragraphs_per_section`: The number of paragraphs to generate per section

Currently, each section has a heading, but individual paragraphs do not have headings. The language model is instructed to distribute the points from the outline across the paragraphs in each section.

Feature Requirement

Implement headings for individual paragraphs to enhance readability and help readers anticipate the content of each paragraph.

Implementation Approaches

Approach 1: Single-Prompt Paragraph and Heading Generation

This approach modifies the existing paragraph generation prompt to request both the paragraph content and an appropriate heading within the same prompt.

Code Changes Required

1. **Modify `prompts.py` - PARAGRAPH_PROMPT**:

```
PARAGRAPH_PROMPT = """
{context_summary}

You are now writing Paragraph {current_paragraph} of
{paragraphs_per_section} for the section titled "{subtitle}" (Section
{section_number} of {total_sections}) in the context of {keyword}.

Requirements:
```

```
1. Generate a short, engaging heading for this paragraph (3-7 words)
2. Generate ONE well-structured paragraph following the heading
3. Focus on a single main point or aspect
4. Use {articlelanguage} language
5. Target the {articleaudience} audience
6. Maintain a {voicetone} tone
7. Use {pointofview} point of view
8. Include specific details and examples
9. Natural keyword integration
10. Clear topic sentence and conclusion
11. MAKE SURE TO RETURN IN THIS FORMAT:
    <h4>Your Paragraph Heading Here</h4>
    <p>Your paragraph content here with <strong>proper HTML
formatting</strong> as needed...</p>
12. DO NOT use any markdown formatting, only proper HTML tags
13. The heading should directly relate to the content in the paragraph
14. Only use <strong> and <em> tags for emphasis

The section needs to cover these points across {paragraphs_per_section}
paragraphs:
{all_points}

For this specific paragraph ({current_paragraph} of
{paragraphs_per_section}), focus on:
{current_points}

This is paragraph {current_paragraph} of {paragraphs_per_section} -
structure your content accordingly.
{flow_instruction}

Write a cohesive paragraph with a relevant heading that educates and
engages the reader while maintaining SEO optimization.
Make sure to make it look as human-like as possible, and please avoid any
hyperbolic language.
"""
```

2. **Modify `content_generator.py` - generate_paragraph method**:

```
# No significant changes needed in this method since we'll handle parsing
in _generate_sections
```

3. **Modify `generator.py` - _generate_sections method**:

```
# Add in _generate_sections method
paragraphs = []
for i in range(self.config.paragraphs_per_section):
    current_paragraph = i + 1
    paragraph_with_heading = self.content_generator.generate_paragraph(
        keyword,
```

```
            heading,
            current_paragraph=current_paragraph,
            paragraphs_per_section=self.config.paragraphs_per_section,
            section_number=section_number,
            total_sections=total_sections,
            section_points=section_points,
            web_context=web_context
        )
        if paragraph_with_heading:
            paragraphs.append(paragraph_with_heading)

    # Combine paragraphs without additional processing
    section_content = "\n\n".join(paragraphs)
```

Approach 2: Sequential Paragraph-then-Heading Generation

This approach first generates the paragraph content and then sends it back to the language model to create a suitable heading.

Code Changes Required

1. **Create a new prompt in `prompts.py`**:

```
PARAGRAPH_HEADING_PROMPT = """
Based on the following paragraph content, generate an appropriate, concise
heading (3-7 words) that accurately represents the main topic or focus of
this paragraph. The heading should be engaging and help readers anticipate
what they will learn.

The heading should be in {articlelanguage} and maintain a {voicetone} tone.

PARAGRAPH CONTENT:
{paragraph_content}

CONTEXT:
- This is paragraph {current_paragraph} of {paragraphs_per_section} in the
section titled "{subtitle}"
- The overall article is about {keyword}

GUIDELINES:
1. Create a brief, descriptive heading (3-7 words)
2. Make it specific to the paragraph content
3. Use action words when appropriate
4. Include keywords naturally if they fit
5. Avoid generic headings like "Introduction" or "Conclusion"
6. DO NOT summarize the entire paragraph, focus on the main point
7. Return ONLY the heading text without any prefix, quotes, or additional
text
"""
```

2. **Add a new method to `content_generator.py`:**

```
def generate_paragraph_heading(self, paragraph_content: str, keyword: str,
subtitle: str,
                               current_paragraph: int = 1,
paragraphs_per_section: int = None) -> str:
    """Generates a heading for a paragraph based on its content."""
    try:
        provider.debug(f"Generating heading for paragraph
{current_paragraph} in section: {subtitle}")

        # Set defaults if not provided
        if paragraphs_per_section is None:
            paragraphs_per_section = self.config.paragraphs_per_section

        # Prepare prompt
        prompt = self.prompts.format_prompt(
            'paragraph_heading',
            paragraph_content=paragraph_content,
            keyword=keyword,
            subtitle=subtitle,
            articlelanguage=self.config.articlelanguage,
            voicetone=self.config.voicetone,
            current_paragraph=current_paragraph,
            paragraphs_per_section=paragraphs_per_section,
        )

        self.context.add_message("user", prompt)
        heading = self._generate_content(
            prompt,
            max_tokens=50,  # Small token limit for a heading
            generation_type="content_generation",
            seed=None
        )

        # Clean up any markdown artifacts and extra whitespace
        heading = heading.replace('*', '').replace('#', '').replace('"',
'').strip()

        provider.debug(f"Generated heading: {heading}")
        return heading
    except Exception as e:
        provider.error(f"Error generating paragraph heading: {str(e)}")
        provider.error(f"Stack trace:\n{traceback.format_exc()}")
        return f"Paragraph {current_paragraph}"
```

3. **Modify the `_generate_sections` method in `generator.py`:**

```
# Inside _generate_sections method
paragraphs_with_headings = []
```

```
for i in range(self.config.paragraphs_per_section):
    current_paragraph = i + 1

    # Generate paragraph content
    paragraph_content = self.content_generator.generate_paragraph(
        keyword,
        heading,
        current_paragraph=current_paragraph,
        paragraphs_per_section=self.config.paragraphs_per_section,
        section_number=section_number,
        total_sections=total_sections,
        section_points=section_points,
        web_context=web_context
    )

    if paragraph_content:
        # Generate heading for this paragraph
        paragraph_heading =
self.content_generator.generate_paragraph_heading(
            paragraph_content=paragraph_content,
            keyword=keyword,
            subtitle=heading,
            current_paragraph=current_paragraph,
            paragraphs_per_section=self.config.paragraphs_per_section
        )

        # Format with HTML heading
        formatted_paragraph = f'<h4>{paragraph_heading}</h4>\n\n<p>
{paragraph_content}</p>'
        paragraphs_with_headings.append(formatted_paragraph)

# Combine paragraphs with their headings
section_content = "\n\n".join(paragraphs_with_headings)
```

Approach 3: Section-Level Heading Generation

This approach provides the entire section to the language model and requests headings for all paragraphs within that section.

Code Changes Required

1. **Create a new prompt in `prompts.py`**:

```
SECTION_PARAGRAPHS_HEADINGS_PROMPT = """
You are given the content of a section with {paragraphs_per_section}
paragraphs. Create appropriate headings for each paragraph while
maintaining coherence across the entire section.

SECTION TITLE: {subtitle}
CONTEXT: This is section {section_number} of {total_sections} in an article
about {keyword}
```

```
SECTION CONTENT:
{section_content}

TASK:
1. Create {paragraphs_per_section} engaging and descriptive headings (3-7
words each)
2. Each heading should accurately represent the content of its
corresponding paragraph
3. Headings should flow logically and complement each other
4. Include keywords naturally where appropriate
5. Use action words and specific terms that engage readers
6. Headings should help readers anticipate the content of each paragraph

RETURN FORMAT:
Return a JSON array with exactly {paragraphs_per_section} headings, in the
same order as the paragraphs. Format:
[
  "Heading for Paragraph 1",
  "Heading for Paragraph 2",
  ...
]

DO NOT include paragraph numbers, additional text, or explanations. ONLY
the JSON array of headings is needed.
"""
```

2. **Add a new method to** `content_generator.py`:

```
def generate_section_paragraph_headings(self, section_content: str,
keyword: str, subtitle: str,
                                        section_number: int = 1,
total_sections: int = 1,
                                        paragraphs_per_section: int = None)
-> List[str]:
    """Generates headings for all paragraphs in a section."""
    try:
        provider.debug(f"Generating headings for paragraphs in section:
{subtitle}")

        # Set defaults if not provided
        if paragraphs_per_section is None:
            paragraphs_per_section = self.config.paragraphs_per_section

        # Prepare prompt
        prompt = self.prompts.format_prompt(
            'section_paragraphs_headings',
            section_content=section_content,
            keyword=keyword,
            subtitle=subtitle,
            section_number=section_number,
            total_sections=total_sections,
```

```
            paragraphs_per_section=paragraphs_per_section,
        )

        self.context.add_message("user", prompt)
        response = self._generate_content(
            prompt,
            max_tokens=200,
            generation_type="content_generation",
            seed=None
        )

        # Parse JSON response
        try:
            # Extract JSON from response if there's additional text
            json_str = re.search(r'\[.*\]', response, re.DOTALL)
            if json_str:
                headings = json.loads(json_str.group(0))
            else:
                headings = json.loads(response)

            # Validate headings count
            if len(headings) != paragraphs_per_section:
                provider.warning(f"Expected {paragraphs_per_section}
headings but got {len(headings)}. Adjusting...")
                # Extend or truncate to match expected count
                if len(headings) < paragraphs_per_section:
                    headings.extend([f"Paragraph {i+1}" for i in
range(len(headings), paragraphs_per_section)])
                else:
                    headings = headings[:paragraphs_per_section]

            return headings
        except json.JSONDecodeError:
            provider.error(f"Failed to parse headings JSON: {response}")
            return [f"Paragraph {i+1}" for i in
range(paragraphs_per_section)]
    except Exception as e:
        provider.error(f"Error generating paragraph headings: {str(e)}")
        provider.error(f"Stack trace:\n{traceback.format_exc()}")
        return [f"Paragraph {i+1}" for i in range(paragraphs_per_section)]
```

3. **Modify the `_generate_sections` method in `generator.py`**:

```
# First generate all paragraphs without headings
paragraphs = []
for i in range(self.config.paragraphs_per_section):
    current_paragraph = i + 1
    paragraph_content = self.content_generator.generate_paragraph(
        keyword,
        heading,
        current_paragraph=current_paragraph,
        paragraphs_per_section=self.config.paragraphs_per_section,
```

```
            section_number=section_number,
            total_sections=total_sections,
            section_points=section_points,
            web_context=web_context
        )
        if paragraph_content:
            paragraphs.append(paragraph_content)

    if not paragraphs:
        logger.error(f"No paragraphs generated for heading: {heading}",
    show_traceback=True)
        return None

    # Combine all paragraphs into raw section content
    raw_section_content = "\n\n".join(paragraphs)

    # Generate headings for all paragraphs in the section
    paragraph_headings =
    self.content_generator.generate_section_paragraph_headings(
        section_content=raw_section_content,
        keyword=keyword,
        subtitle=heading,
        section_number=section_number,
        total_sections=total_sections,
        paragraphs_per_section=self.config.paragraphs_per_section
    )

    # Combine paragraphs with their headings
    paragraphs_with_headings = []
    for i, (paragraph, heading) in enumerate(zip(paragraphs,
    paragraph_headings)):
        formatted_paragraph = f'<h4>{heading}</h4>\n\n<p>{paragraph}</p>'
        paragraphs_with_headings.append(formatted_paragraph)

    # Final section content with paragraph headings
    section_content = "\n\n".join(paragraphs_with_headings)
```

Approach 4: Outline-Based Paragraph Headings

This approach involves using the outline points as paragraph headings. We'll define a variable number of subheadings (0-5) for each main heading.

Code Changes Required

1. **Modify `config.py` to add a new parameter**:

```
@dataclass
class Config:
    # Existing parameters...

    # Paragraph Heading Settings
```

```
    enable_paragraph_headings: bool = True
    max_paragraph_headings_per_section: int = 5  # Maximum number of
paragraph headings per section
```

2. **Modify the outline prompt in `prompts.py`:**

```
OUTLINE_PROMPT = """
===== CRITICAL FORMATTING REQUIREMENTS =====
You MUST create an outline that EXACTLY follows this format with NO
DEVIATIONS:

I. [Main Section Title]
A. [Subsection Point - Will be used as paragraph heading]
B. [Subsection Point - Will be used as paragraph heading]
C. [Subsection Point - Will be used as paragraph heading]

II. [Main Section Title]
A. [Subsection Point - Will be used as paragraph heading]
B. [Subsection Point - Will be used as paragraph heading]
C. [Subsection Point - Will be used as paragraph heading]

===== STRICT STRUCTURAL RULES =====
• EXACTLY {sizesections} main sections (numbered with Roman numerals I.,
II., III., etc.)
• EXACTLY {sizeheadings} subsections per main section (lettered A., B., C.,
etc.)
• The subsection points will be used as paragraph headings
• EXACTLY one blank line between each main section
• NO Introduction or Conclusion sections in the outline
• NO extra text, explanations, or notes outside the outline format
• NO bullet points, numbered lists, or any other formatting

===== CONTENT GUIDELINES =====
• Main section titles: Clear, descriptive phrases about {keyword} (5-10
words)
• Subsection points: Specific aspects that will serve as paragraph headings
(5-10 words each)
• Write subsection points as concise headings, not complete sentences
• Each subsection point should be distinct and cover a different aspect
• Content should be appropriate for {articleaudience} audience level
• Content should follow {articletype} style and approach
• All content must directly relate to {keyword}
"""
```

3. **Modify the `_generate_sections` method in `generator.py`:**

```
# Inside _generate_sections method, after extracting section_points
paragraph_headings = section_points.copy()
```

```
    # If we have more headings than paragraphs, truncate
    if len(paragraph_headings) > self.config.paragraphs_per_section:
        paragraph_headings =
    paragraph_headings[:self.config.paragraphs_per_section]

    # If we have fewer headings than paragraphs, generate generic ones for the
    rest
    while len(paragraph_headings) < self.config.paragraphs_per_section:
        paragraph_headings.append(f"Additional Information on {heading}")

    # Generate paragraphs
    paragraphs = []
    for i in range(self.config.paragraphs_per_section):
        current_paragraph = i + 1
        current_heading = paragraph_headings[i] if i < len(paragraph_headings)
    else f"Paragraph {current_paragraph}"

        paragraph_content = self.content_generator.generate_paragraph(
            keyword,
            heading,
            current_paragraph=current_paragraph,
            paragraphs_per_section=self.config.paragraphs_per_section,
            section_number=section_number,
            total_sections=total_sections,
            section_points=section_points,
            web_context=web_context
        )

        if paragraph_content:
            # Format with HTML heading
            formatted_paragraph = f'<h4>{current_heading}</h4>\n\n<p>
    {paragraph_content}</p>'
            paragraphs.append(formatted_paragraph)

    # Combine paragraphs with their headings
    section_content = "\n\n".join(paragraphs)
```

Implementation Recommendations

Primary Recommendation: Hybrid Approach (Combining Approach 4 and Approach 1)

This hybrid approach combines the strengths of both the Outline-Based Headings and Single-Prompt Generation approaches, offering the best of both worlds. It uses outline points as initial paragraph headings but also gives the language model an opportunity to refine these headings during paragraph generation.

Code Changes Required

1. **Modify `config.py` to add configuration parameters**:

```
    @dataclass
    class Config:
```

```
    # Existing parameters...

    # Paragraph Heading Settings
    enable_paragraph_headings: bool = True
    max_paragraph_headings_per_section: int = 5  # Maximum number of
paragraph headings per section
    refine_paragraph_headings: bool = True  # Whether to allow the LLM to
refine outline-based headings
```

2. **Modify `prompts.py` - PARAGRAPH_PROMPT with heading refinement**:

```
PARAGRAPH_PROMPT = """
{context_summary}

You are now writing Paragraph {current_paragraph} of
{paragraphs_per_section} for the section titled "{subtitle}" (Section
{section_number} of {total_sections}) in the context of {keyword}.

SUGGESTED HEADING: {suggested_heading}

Requirements:
1. Consider the suggested heading above. You may use it as-is or refine it
to better match your paragraph content
2. Generate ONE well-structured paragraph
3. Focus on a single main point or aspect
4. Use {articlelanguage} language
5. Target the {articleaudience} audience
6. Maintain a {voicetone} tone
7. Use {pointofview} point of view
8. Include specific details and examples
9. Natural keyword integration
10. Clear topic sentence and conclusion
11. MAKE SURE TO RETURN IN THIS FORMAT:
    <h4>Your Final Paragraph Heading Here</h4>
    <p>Your paragraph content here with <strong>proper HTML
formatting</strong> as needed...</p>
12. DO NOT use any markdown formatting, only proper HTML tags
13. The heading should directly relate to the content in the paragraph
14. Only use <strong> and <em> tags for emphasis

The section needs to cover these points across {paragraphs_per_section}
paragraphs:
{all_points}

For this specific paragraph ({current_paragraph} of
{paragraphs_per_section}), focus on:
{current_points}

This is paragraph {current_paragraph} of {paragraphs_per_section} -
structure your content accordingly.
{flow_instruction}
```

```
Write a cohesive paragraph with a relevant heading that educates and
engages the reader while maintaining SEO optimization.
Make sure to make it look as human-like as possible, and please avoid any
hyperbolic language.
"""
```

3. **Modify the `generate_paragraph` method in `content_generator.py`**:

```
def generate_paragraph(self, keyword: str, subtitle: str,
current_paragraph: int = 1,
                       paragraphs_per_section: int = None, section_number:
int = 1,
                       total_sections: int = 1, section_points: List[str] =
None,
                       suggested_heading: str = None, web_context: str = "")
-> str:
    """Generates a paragraph for a specific subtitle with explicit
positioning."""
    try:
        provider.debug(f"Generating paragraph
{current_paragraph}/{paragraphs_per_section} for: {subtitle}")

        # Set defaults if not provided
        if paragraphs_per_section is None:
            paragraphs_per_section = self.config.paragraphs_per_section

        # Default empty list for section points if none provided
        if section_points is None:
            section_points = []

        # If no suggested heading is provided, use a generic one
        if suggested_heading is None:
            suggested_heading = f"Information about {subtitle}"

        # Distribute points across paragraphs if there are enough points
        # ... [existing distribution code] ...

        # Prepare prompt with suggested heading
        prompt = self.prompts.format_prompt(
            'paragraph',
            context_summary=self.context.get_context_summary(),
            keyword=keyword,
            subtitle=subtitle,
            suggested_heading=suggested_heading,  # Pass the suggested
heading
            articlelanguage=self.config.articlelanguage,
            # ... [other existing parameters] ...
        )

        # ... [rest of the method remains the same] ...
    except Exception as e:
        provider.error(f"Error generating paragraph: {str(e)}")
```

```
            provider.error(f"Stack trace:\n{traceback.format_exc()}")
            return ""
```

4. **Modify the `_generate_sections` method in `generator.py`**:

```
# Inside _generate_sections method, after extracting section_points
paragraph_headings = section_points.copy()

# If we have more headings than paragraphs, truncate
if len(paragraph_headings) > self.config.paragraphs_per_section:
    paragraph_headings =
paragraph_headings[:self.config.paragraphs_per_section]

# If we have fewer headings than paragraphs, generate generic ones for the
rest
while len(paragraph_headings) < self.config.paragraphs_per_section:
    paragraph_headings.append(f"Additional Information on {heading}")

# Generate paragraphs with suggested headings
paragraphs = []
for i in range(self.config.paragraphs_per_section):
    current_paragraph = i + 1
    suggested_heading = paragraph_headings[i] if i <
len(paragraph_headings) else f"Paragraph {current_paragraph}"

    # Pass the suggested heading to the paragraph generator
    paragraph_with_heading = self.content_generator.generate_paragraph(
        keyword,
        heading,
        current_paragraph=current_paragraph,
        paragraphs_per_section=self.config.paragraphs_per_section,
        section_number=section_number,
        total_sections=total_sections,
        section_points=section_points,
        suggested_heading=suggested_heading,  # Pass the suggested heading
        web_context=web_context
    )

    if paragraph_with_heading:
        paragraphs.append(paragraph_with_heading)

# Combine paragraphs without additional processing
section_content = "\n\n".join(paragraphs)
```

This hybrid approach offers several advantages:

- Uses outline points as initial headings, maintaining structural consistency
- Allows the LLM to refine headings during paragraph generation for better coherence
- Single API call per paragraph (efficient)
- Headings remain contextually relevant to actual content

- Preserves the hierarchical relationship between sections, subsections, and paragraphs

Alternative Recommendation 1: Approach 1 (Single-Prompt Generation)

This approach is the most efficient in terms of API calls and processing time, as it generates both the heading and paragraph content in a single operation. It also ensures the best coherence between the heading and content since they're generated together.

Alternative Recommendation 2: Approach 4 (Outline-Based Headings)

This approach aligns well with the client's suggestion to use a variable number of subheadings. By using the outline points as paragraph headings, we ensure that the article structure is consistent from planning to final output.

Considerations for Implementation

1. **HTML Formatting**:
    - Ensure consistent heading levels are used (e.g., `<h4>` for paragraph headings)
    - Maintain proper HTML structure in generated content
2. **Configuration Parameter**:
    - Add `enable_paragraph_headings` parameter to allow toggling this feature
3. **Output Formatting**:
    - Update the Markdown and HTML output formats to properly include paragraph headings
    - Ensure heading styles are consistent with the rest of the document
4. **User Testing**:
    - Implement in a way that allows for easy comparison of articles with and without paragraph headings
    - Gather feedback on readability improvements

Additional Enhancement: Variable Number of Paragraph Headings

To implement the client's suggestion of variable paragraph headings (0-5 per main heading):

1. Add a new configuration parameter:

```
variable_paragraph_headings: bool = True
```

2. Modify the content generation to dynamically determine the number of headings:

```
# If variable_paragraph_headings is enabled, use between 0-5 headings per
section
if self.config.variable_paragraph_headings:
    # Randomly determine number of headings for this section (between 0 and
```

```
max_paragraph_headings_per_section)
    import random
    num_headings = random.randint(0,
min(self.config.max_paragraph_headings_per_section,
self.config.paragraphs_per_section))

    # If num_headings is less than paragraphs_per_section, some paragraphs
won't have headings
    paragraph_headings = section_points[:num_headings]
    paragraph_headings.extend([None] * (self.config.paragraphs_per_section
- num_headings))
```

3. Then in the paragraph generation loop:

```
for i in range(self.config.paragraphs_per_section):
    current_paragraph = i + 1
    current_heading = paragraph_headings[i]

    # Generate paragraph content...

    # Only add heading if one exists for this paragraph
    if current_heading:
        formatted_paragraph = f'<h4>{current_heading}</h4>\n\n<p>
{paragraph_content}</p>'
    else:
        formatted_paragraph = f'<p>{paragraph_content}</p>'
```

This enhancement would allow for natural variety in the content structure while still maintaining the overall organization of the article.