```
# Python code to turn WORD paragraphs into chunks for embeddings
# Provides three options for the chunking.
# see function at end; def get_optimized_chunks(path_to_doc: str | Path) ->
List[str]:
#  that selects a technique based on the properties of the Word document

from typing import List, Optional
import re
from docx import Document
from pathlib import Path

def process_chunks_simple(paragraphs: List[str],
                          min_length: int = 50,
                          max_length: int = 1000) -> List[str]:
    """
    Basic chunk processing with length filters.

    Args:
        paragraphs: List of paragraph strings
        min_length: Minimum characters per chunk
        max_length: Maximum characters per chunk

    Returns:
        List of processed chunks
    """
    processed_chunks = []
    current_chunk = ""

    for para in paragraphs:
        para = para.strip()

        # Skip if too short
        if len(para) < min_length:
            continue

        # Split if too long
        if len(para) > max_length:
            # Split on sentences
            sentences = re.split(r'(?<=[.!?])\s+', para)
            for sentence in sentences:
                if len(sentence) >= min length:
                    processed_chunks.append(sentence)
        else:
            processed_chunks.append(para)

    return processed_chunks

def process_chunks_advanced(paragraphs: List[str],
                            target_size: int = 500,
                            overlap: int = 50,
                            min_size: int = 100) -> List[str]:
    """
    Advanced chunk processing with overlap and smart merging.

    Args:
        paragraphs: List of paragraph strings
        target_size: Target chunk size in characters
        overlap: Number of characters to overlap between chunks
        min_size: Minimum chunk size

    Returns:
        List of processed chunks
    """
    chunks = []
```

```
    current_chunk = ""

    for para in paragraphs:
        para = para.strip()

        # If paragraph is very long, split it into sentences
        if len(para) > target_size:
            sentences = re.split(r'(?<=[.!?])\s+', para)

            for sentence in sentences:
                if len(current_chunk) + len(sentence) <= target_size:
                    current_chunk += (" " + sentence if current_chunk else
                    sentence)
                else:
                    if len(current_chunk) >= min_size:
                        chunks.append(current_chunk)

                    # Start new chunk with overlap
                    if len(current_chunk) > overlap:
                        words = current_chunk.split()
                        overlap_text = " ".join(words[-overlap:])
                        current_chunk = overlap_text + " " + sentence
                    else:
                        current_chunk = sentence
        else:
            # Handle regular paragraphs
            if len(current_chunk) + len(para) <= target_size:
                current_chunk += (" " + para if current_chunk else para)
            else:
                if len(current_chunk) >= min_size:
                    chunks.append(current_chunk)
                current_chunk = para

    # Add the last chunk if it meets minimum size
    if len(current chunk) >= min size:
        chunks.append(current_chunk)

    return chunks

def process_chunks_semantic(paragraphs: List[str],
                            max tokens: int = 500,
                            preserve_sentences: bool = True) -> List[str]:
    """
    Semantic-aware chunk processing that preserves context.

    Args:
        paragraphs: List of paragraph strings
        max_tokens: Maximum tokens per chunk (approximate)
        preserve_sentences: Whether to avoid splitting sentences

    Returns:
        List of processed chunks
    """
    def estimate_tokens(text: str) -> int:
        # Rough estimation: words / 0.75 (assuming average word length)
        return len(text.split()) // 0.75

    chunks = []
    current_chunk = []
    current_token_count = 0

    for para in paragraphs:
        sentences = re.split(r'(?<=[.!?])\s+', para.strip()) if
        preserve_sentences else [para]
```

```
        for sentence in sentences:
            sentence_tokens = estimate_tokens(sentence)

            if sentence_tokens > max_tokens:
                # Handle very long sentences if we're not preserving them
                if not preserve_sentences:
                    words = sentence.split()
                    current_words = []

                    for word in words:
                        current_words.append(word)
                        if estimate_tokens(" ".join(current_words)) >=
                        max_tokens:
                            chunks.append(" ".join(current_words))
                            current_words = []

                    if current_words:
                        chunks.append(" ".join(current_words))
                continue

            if current_token_count + sentence_tokens <= max_tokens:
                current_chunk.append(sentence)
                current_token_count += sentence_tokens
            else:
                if current_chunk:
                    chunks.append(" ".join(current_chunk))
                current_chunk = [sentence]
                current_token_count = sentence_tokens

    if current_chunk:
        chunks.append(" ".join(current_chunk))

    return chunks

# Example usage combining all approaches
def get_optimized_chunks(path_to_doc: str | Path) -> List[str]:
    """
    Extract and optimize text chunks from a Word document.
    """
    doc = Document(path to doc)
    paragraphs = [p.text.strip() for p in doc.paragraphs if p.text.strip()]

    # Process with different strategies based on content
    if len(paragraphs) < 10:  # Short document
        return process_chunks_simple(paragraphs)
    elif any(len(p) > 1000 for p in paragraphs):  # Contains very long
    paragraphs
        return process_chunks_advanced(paragraphs)
    else:  # Normal document
        return process_chunks_semantic(paragraphs)
```