What is AutoGen?

Agentic chunking is the use of artificial intelligence (AI) to segment lengthy text inputs into smaller, semantically coherent blocks known as chunks. While many traditional chunking strategies tend to use fixed-size chunks when splitting text, agentic chunking dynamically segments text based on context.

Large language models (LLMs) cannot process a large text-based input sequence in its entirety. A natural language processing (NLP) model's context window determines the maximum amount of content the model can intake while maintaining contextual understanding. Machine learning (ML) systems use chunking techniques to break documents into pieces that fit the LLM's context window.

Your subscription will be delivered in English. You will find an unsubscribe link in every newsletter. You can manage your subscriptions or unsubscribe here. Refer to our IBM Privacy Statement for more information.

Chunking and RAG

The development of retrieval-augmented generation (RAG)—connecting LLMs to external data sources—necessitated the creation of chunking systems. RAG systems emerged to help counter the problem of hallucinations: when LLMs would deliver answers that didn't reflect real-world outcomes or information.

RAG systems help LLMs generate more accurate, more useful answers by pairing them with additional knowledge bases. In many cases, RAG knowledge bases are vector databases containing documents that give the connected LLM access to domain-specific knowledge. Embedding models convert documents into mathematical vectors, then do the same for user queries.

The RAG system finds embeddings within its vector database that represent relevant information and match the user query. Then, the LLM uses the retrieved data to provide users with more relevant and accurate answers.

But due to the limitations of the context window, the LLM isn't able to process a single document at once. Chunking emerged as the solution. By breaking a document into pieces, the LLM can efficiently find relevant chunks in real time while maintaining contextual understanding.

Other chunking methods

Agentic chunking allows LLMs to create meaningful chunks that help them provide better answers, such as with the RAG use case. Some chunking methods also take semantics into account, while others break documents into smaller chunks of fixed length.

Other chunking methods include:

Fixed-size chunking

The simplest chunking strategy, fixed-size chunking splits text into blocks of the same size, based on a preset character or token count. A token is the minimum amount of text that the LLM can process—often a word or a portion of one.

To avoid breaking up sentences, many fixed-size chunking implementations include an overlap feature that repeats the end of one chunk at the beginning of the next. Fixed-size chunking is simple and computationally light, but is rigid—it can't account for content density or document structure and can create semantically incoherent chunks.

Recursive chunking

Recursive chunking uses a hierarchical list of predefined text separators to break text up in a way that is likely to be more coherent. Separators include naturally occurring structures such as paragraphs, sentences or words. In a Python coding document, separators can include class and function definitions.

Compared to fixed-size chunking, recursive chunking creates more coherent chunks by following naturally occurring separations in the text. The use of markdown can also help the chunking algorithm, or chunker, figure out where to make divisions. *RecursiveCharacterTextSplitter* is a popular chunker available in LangChain.

But if the text lacks clear separators, the recursive chunking algorithms won't know where to create new chunks. Recursive chunking is also more computationally intensive than fixed-size chunking.

Semantic chunking

Semantic chunking uses embedding models to create mathematical representations of each sentence. Then, the chunking algorithm creates chunks of semantically similar sentences, creating a new chunk is created it detects a change in semantics. Semantic chunking is attributed to Greg Kamradt, who discussed the technique on Github.[1]

Semantic chunking is context-aware, building chunks around the natural flow and semantic meaning of the document. When the topic changes, a new chunk is created. However, issues can emerge when paragraphs discuss multiple topics or if the chunking threshold isn't set properly for the document type and structure.

Semantic chunking is more computationally intensive than recursive and fixed-size chunking and requires advanced models to identify semantic content within the text.

How does agentic chunking work?

Agentic chunking is an example of agentic automation: using AI agents to automate a workflow. In this case, the workflow being intelligently automated is the process of determining how to split a document into smaller chunks that fit within an LLM's context window.

Agentic AI refers to the use of AI systems to make autonomous decisions and take action without human intervention. With agentic chunking, the agent acts on its own to determine how to separate the text and label the chunks.

Agentic chunking draws from other chunking methods to create overlapping sections and recursive splitting, then applies generative AI (gen AI) to label each chunk with metadata for easier RAG retrieval.

Agentic chunking is still in the exploratory stages. Creators share and discuss their approaches on GitHub. These are often built with the Python coding language using LLM frameworks such as Llamaindex and Langchain along with open source LLMs available on Huggingface.

A typical AI workflow for agentic chunking might contain these steps:

1. Text preparation

Using intelligent automation tools, the text is extracted from the source document, such as a PDF, and cleaned. Text cleaning involves the removal of superfluous elements such as page numbers and footers so that the LLM is given only raw text.

2. Text splitting

Recursive chunking algorithms split the text into small chunks to avoid chopping sentences into pieces. Like semantic chunking, agentic chunking dynamically splits text based on meaning, structure and context awareness using the chunk overlap technique.

3. Chunk labeling

LLMs, such as OpenAI's GPT, process, combine and enrich the chunks. Smaller chunks are combined into larger ones that maintain semantic coherence. The LLM enriches each chunk with metadata that includes a title and summary of the chunk's contents. The generated metadata assists downstream uses such as with agentic RAG systems.

4. Embedding

Each chunk is converted into an embedding and stored in a vector database. Retrieval models query the database, use semantic search to find chunks with relevant metadata, and include them in the prompts for the LLM in the RAG system.

The prompt_template setting in LangChain determines the input prompt given to the LLM. Learn more about how to optimize RAG chunking with LangChain and watsonx.ai.

Benefits of agentic chunking

Compared to traditional chunking methods, agentic chunking’s dynamism and metadata labeling makes it a great fit for RAG implementation. The benefits include:

- **Efficient retrieval:** The AI-generated titles and summaries for each chunk can help RAG systems find relevant information faster in connected datasets.

- **Accurate responses:** Semantically coherent chunking with AI-generated metadata can aid RAG systems in augmenting generated responses with relevant data for better answers.

- **Flexibility:** AI-driven chunking can handle a wide range of document types. Agentic chunking systems can integrate with various LLMs and RAG chains to keep pace with project growth and expansion.

- **Content preservation:** Agentic chunking systems build on earlier chunking methods to create chunks that preserve semantic meaning and coherency.