Chunking Methods and Their Relation to Context Windows

AI Research Team

May 14, 2025

Abstract

This article explores the relationship between chunking methods and context windows in large language models (LLMs). We provide a comprehensive overview of various chunking strategies, their mathematical foundations, and practical applications. The article discusses how effective chunking techniques can optimize the use of context windows, leading to improved performance in tasks such as question answering, summarization, and search. We also examine the algorithmic details of different chunking methods and their implementation considerations.

1 Introduction

Large language models (LLMs) have revolutionized natural language processing, but they come with inherent limitations, particularly in the form of context windows. A context window refers to the maximum amount of text an LLM can process at once, representing the model's "short-term memory." As these models continue to advance, understanding and optimizing the use of context windows becomes increasingly important.

Chunking strategies provide a solution to this limitation by breaking down long or complex text into smaller, manageable parts called "chunks." This approach helps LLMs process information more effectively without missing important details or context. In this article, we explore the relationship between chunking methods and context windows, examining how different chunking strategies can be employed to maximize the effectiveness of LLMs.

2 Context Windows: The Foundation

2.1 Definition and Purpose

A context window is an AI's 'short-term memory,' allowing it to give more tailored responses based on an ongoing conversation or uploaded documents. It represents the number of tokens a model can consider when responding to

prompts and inputs and functions as the AI's "working memory" for a particular analysis or conversation.

Mathematically, we can define a context window W as:

$$W = \{t_1, t_2, \ldots, t_n\} \quad \text{where} \quad n \leq N_{max} \tag{1}$$

Here, t_i represents the i-th token in the sequence, and N_{max} is the maximum number of tokens the model can process at once.

2.2 Tokenization Process

Tokenization is a crucial step in language model processing. It involves breaking down unstructured text into manageable units called tokens. These tokens can be words, characters, or even pieces of words, serving as the fundamental building blocks that algorithms use to understand text.

The tokenization process employs various algorithms, such as WordPiece or Byte Pair Encoding (BPE). For a text T, the tokenization function tokenize(T) produces a sequence of tokens:

$$\text{tokenize}(T) = [t_1, t_2, \ldots, t_m] \tag{2}$$

Generally, one token corresponds to about 4 characters of English text, which is approximately $\frac{3}{4}$ of a word.

2.3 Importance in Language Models

Context windows are vital in determining a model's ability to make coherent and contextually relevant responses or analyses. The size of the context window significantly impacts the model's performance. A larger context window offers a broader view, empowering the model to capture longer-range dependencies and nuances.

However, increasing the context window size in traditional transformer-based models can be challenging. As the context window grows linearly, the number of model parameters increases quadratically, leading to complexities in scaling.

2.4 Current Context Window Sizes

Different LLMs offer varying context window sizes:

- ChatGPT: 128,000 tokens
- Google Gemini: 1,000,000 tokens
- Claude: 200,000 tokens
- Microsoft Copilot: 128,000 tokens
- Mistral: 32,000 tokens

3 Chunking Strategies: An Overview

Chunking strategies are essential when working with large language models because they determine how information is divided before being processed. A well-designed strategy helps preserve meaning, context, and relevance in each segment, leading to better outputs.

3.1 Why We Need Chunking

In language processing with LLMs, using "chunks" is necessary to handle long pieces of text effectively. LLMs have a limit on how much text they can process at once. If the input exceeds that limit, the model may miss important information or stop processing entirely.

Chunking ensures that each piece of information gets the attention it needs, improving response speed, especially when combined with search and retrieval techniques.

3.2 Types of Chunking Methods

There are several approaches to chunking text for LLM processing:

3.2.1 Fixed-size Chunking

Fixed-size chunking breaks content by a set number of tokens or characters. It's ideal for uniform processing but may cut off context. Mathematically, for a text T with tokens $[t_1, t_2, \ldots, t_m]$ and a fixed chunk size k, the chunks C_i are defined as:

$$C_i = [t_{(i-1)\cdot k+1}, t_{(i-1)\cdot k+2}, \ldots, t_{i\cdot k}] \quad \text{for} \quad i = 1, 2, \ldots, \lceil m/k \rceil \tag{3}$$

3.2.2 Semantic Chunking

Semantic chunking uses natural breaks such as paragraphs, bullet points, or sections. It maintains context but may vary in chunk size. Let $S = \{s_1, s_2, \ldots, s_p\}$ be the set of semantic boundaries in the text. The chunks C_j are defined as:

$$C_j = [t_{s_j+1}, t_{s_j+2}, \ldots, t_{s_{j+1}}] \quad \text{for} \quad j = 0, 1, \ldots, p-1 \tag{4}$$

where $s_0 = 0$ and $s_p = m$.

3.2.3 Hybrid Chunking

Hybrid chunking combines both methods. For example, use semantic chunking with a token limit to keep chunks both meaningful and manageable. This approach can be represented as:

$$C_j = \min(\text{SemanticChunk}_j, \text{FixedSizeChunk}_j) \tag{5}$$

where the minimum operation ensures that chunks don't exceed a certain size while still respecting semantic boundaries when possible.

3.2.4 Overlapping Chunks

In some cases, adding overlap between chunks improves context retention. If O is the overlap size, the overlapping chunks C_i^O are defined as:

$$C_i^O = [t_{(i-1)\cdot(k-O)+1}, t_{(i-1)\cdot(k-O)+2}, \ldots, t_{(i-1)\cdot(k-O)+k}] \tag{6}$$

This approach is especially useful for narrative content or when dealing with FAQs, instructions, or knowledge bases where continuity matters.

4 Advanced Chunking Techniques

4.1 Dynamic Chunking Based on Query Intent

Dynamic chunking tailors the chunks in real-time based on the user's query. It extracts content that is most relevant, slices it intelligently around the query context, and sends it to the LLM.

Given a query q and a document D, the dynamic chunking function $\text{DynamicChunk}(q, D)$ produces chunks that are most relevant to the query:

$$C_q = \text{DynamicChunk}(q, D) = \underset{C \subset D}{\text{argmax}} \text{Relevance}(q, C) \tag{7}$$

where $\text{Relevance}(q, C)$ measures the semantic similarity between the query and the chunk.

4.2 Hierarchical Chunking

Hierarchical chunking involves creating multiple levels of chunks—for example, section ¿ paragraph ¿ sentence. Based on the query or use case, the application can choose which level of chunking to use.

For a document D, the hierarchical chunking function $\text{HierarchicalChunk}(D)$ produces a nested structure of chunks:

$$\text{HierarchicalChunk}(D) = \{C^1, C^2, \ldots, C^L\} \tag{8}$$

where $C^l = \{C_1^l, C_2^l, \ldots, C_{n_l}^l\}$ represents the chunks at level l, and L is the number of levels.

4.3 Window Sliding with Variable Overlap

Instead of using fixed overlapping chunks, this method adjusts the amount of overlap dynamically. For instance, more overlap is applied where content is dense with information or where narrative continuity is essential.

For a text with tokens $[t_1, t_2, \ldots, t_m]$, the variable overlap function VarOverlap(i) determines the overlap size for the i-th chunk:

$$C_i^{\text{var}} = [t_{s_i}, t_{s_i+1}, \ldots, t_{s_i+k-1}] \tag{9}$$

where $s_i = (i-1) \cdot k - \sum_{j=1}^{i-1} \text{VarOverlap}(j)$ is the starting position of the i-th chunk.

4.4 Embedding-Aware Chunking

Embedding-aware chunking leverages pre-trained embedding models to segment content based on semantic shifts. It analyzes the text for changes in topic or tone and breaks chunks at those points.

For a text with tokens $[t_1, t_2, \ldots, t_m]$ and an embedding function embed(t), the semantic shift at position i can be measured as:

$$\text{Shift}(i) = \text{Distance}(\text{embed}(t_i), \text{embed}(t_{i+1})) \tag{10}$$

Chunks are then created at positions where Shift(i) exceeds a threshold θ.

4.5 Metadata-Driven Chunk Enrichment

Chunks can be enhanced by attaching metadata such as author, date, source, category, and confidence score. This metadata helps retrieval systems filter or prioritize relevant chunks before they reach the LLM.

A chunk with metadata can be represented as a tuple (C, M), where C is the content and $M = \{(k_1, v_1), (k_2, v_2), \ldots, (k_r, v_r)\}$ is the set of metadata key-value pairs.

4.6 Multi-Document Chunking

In complex applications, queries may require information from multiple documents. Multi-document chunking stitches together related chunks from different sources to provide a holistic response.

Given a set of documents $\{D_1, D_2, \ldots, D_p\}$ and a query q, the multi-document chunking function produces a set of relevant chunks across all documents:

$$C_q^{\text{multi}} = \bigcup_{j=1}^{p} \text{DynamicChunk}(q, D_j) \tag{11}$$

5 Architectural Approaches for Chunking

When designing a chunking system for LLM applications, choosing the right architecture is key to maintaining performance, scalability, and accuracy. There are generally two architectural approaches: pre-processing-based chunking and on-the-fly chunking.

5.1 Pre-processing-based Chunking

Pre-processing-based chunking involves preparing and storing the chunks ahead of time. This is ideal for static documents like manuals, knowledge bases, or FAQs. The process can be described algorithmically as follows:

Require: Document collection $D = \{D_1, D_2, \ldots, D_n\}$, Chunking method C
Ensure: Chunk database DB
$DB \leftarrow \emptyset$
for each document D_i in D **do**
  $chunks_i \leftarrow C(D_i)$
  for each chunk c in $chunks_i$ **do**
    $embedding \leftarrow \text{embed}(c)$
    $metadata \leftarrow \text{extractMetadata}(c, D_i)$
    $DB \leftarrow DB \cup \{(c, embedding, metadata)\}$
  end for
end for
return DB

5.2 On-the-fly Chunking

On-the-fly chunking generates chunks in real-time based on user input or dynamic content. It's best for live data such as chat logs, emails, or real-time transcripts. The process can be described as follows:

Require: User query q, Document D, Chunking method C
Ensure: Relevant chunks R
$chunks \leftarrow C(D)$
$R \leftarrow \emptyset$
for each chunk c in $chunks$ **do**
  $relevance \leftarrow \text{computeRelevance}(q, c)$
  if $relevance > threshold$ **then**
    $R \leftarrow R \cup \{c\}$
  end if
end for
return R

6 Implementing Chunking Strategies: Step-by-Step

Successfully implementing a chunking strategy is key to building a high-performing LLM application. Here's a step-by-step guide to help you implement chunking effectively:

6.1 Define Your Objective

Start by identifying the goal of your LLM application. Your use case determines how your chunking should be structured—whether chunks need to be concise and focused or broad and context-rich.

6.2 Analyze Your Data Source

Review the type and format of your content. Structured content lends itself to semantic chunking, where you can split based on headings or sections. Unstructured text may require splitting by sentences or token count.

6.3 Choose the Right Chunking Method

Select from fixed-size chunking, semantic chunking, or hybrid chunking based on your content and objectives.

6.4 Use Token Counter Tools

Since LLMs process text by tokens, use a token counter to ensure each chunk stays within the model's token limit. Leave room for the prompt and response—usually keep chunks under 75% of the total limit.

6.5 Add Overlap for Context (If Needed)

If your content is sequential, include a small overlap (e.g., 10-15% of the previous chunk) in the next chunk. This helps maintain continuity and prevents context loss.

6.6 Store and Index Chunks

If your application includes search or retrieval, store each chunk with relevant metadata such as title, section ID, or tags. This improves search accuracy and speeds up query resolution.

6.7 Integrate with LLM Query System

When a user sends a query, retrieve the most relevant chunk(s) based on keyword or semantic similarity. Pass these chunks to the LLM along with the user prompt.

6.8 Test and Fine-Tune

Test your setup with real-world data. Measure response accuracy, context retention, and speed. Gather user feedback to identify if the chunking approach needs adjustments.

7 Topic Modeling and Its Relation to Chunking

Topic modeling is a technique used to identify the main themes or topics present in a collection of textual data. It can be used in conjunction with chunking to improve the relevance of retrieved chunks.

7.1 Bag of Words

Bag of Words (BoW) is a common representation used in NLP for textual data. It counts the frequency at which each word occurs in a document. For a document D with vocabulary V, the BoW representation is a vector $\vec{b} = [b_1, b_2, \ldots, b_{|V|}]$ where b_i is the frequency of the i-th word in the vocabulary.

7.2 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is a popular probabilistic model used for topic modeling. It is based on the assumption that documents are mixtures of topics, and topics are mixtures of words.

In LDA, a document d is represented as a probability distribution over topics $\vec{\theta}_d = [\theta_{d,1}, \theta_{d,2}, \ldots, \theta_{d,K}]$, where K is the number of topics. Each topic k is represented as a probability distribution over words $\vec{\phi}_k = [\phi_{k,1}, \phi_{k,2}, \ldots, \phi_{k,|V|}]$.

The generative process for LDA can be described as follows:

1. For each topic k, draw a word distribution $\vec{\phi}_k \sim \text{Dirichlet}(\vec{\beta})$
2. For each document d, draw a topic distribution $\vec{\theta}_d \sim \text{Dirichlet}(\vec{\alpha})$
3. For each word position i in document d:
   (a) Draw a topic assignment $z_{d,i} \sim \text{Multinomial}(\vec{\theta}_d)$
   (b) Draw a word $w_{d,i} \sim \text{Multinomial}(\vec{\phi}_{z_{d,i}})$

7.3 Non-Negative Matrix Factorization

Non-Negative Matrix Factorization (NMF) is another topic modeling technique that uncovers latent topics in a collection of documents. It relies on the Term Frequency-Inverse Document Frequency (TF-IDF) representation to capture and retrieve hidden themes or topics from the documents.

Given a TF-IDF matrix $X \in \mathbb{R}^{n \times m}$ where n is the number of documents and m is the vocabulary size, NMF factorizes X into two non-negative matrices $W \in \mathbb{R}^{n \times k}$ and $H \in \mathbb{R}^{k \times m}$ such that $X \approx WH$, where k is the number of topics.

The factorization is typically done by minimizing the Frobenius norm of the difference:

$$\min_{W,H} ||X - WH||_F^2 \quad \text{subject to} \quad W, H \geq 0 \tag{12}$$

8 Real-World Use Cases

Chunking strategies are essential in making large language model (LLM) applications more useful, context-aware, and scalable. Here are some practical use cases across industries:

8.1 Customer Support Knowledge Base

A company integrates an AI chatbot to help users find answers in product manuals and help documents. The content is chunked by section titles and token limits. Each chunk is embedded with metadata like product name, version, and category, and stored in a vector database.

8.2 Legal Document Summarization

A law firm uses an AI tool to summarize long contracts, terms, and case files. Documents are chunked hierarchically—by section, paragraph, and clause. Each chunk is labeled with legal terms or themes (e.g., "termination clause," "confidentiality").

8.3 Academic Research Assistant

Students and researchers use an LLM-powered assistant to review academic papers. Research papers are chunked by abstract, methods, results, and discussion sections. Semantic chunking is used to retain topic continuity.

8.4 Internal Enterprise Search

Large organizations implement AI search tools for employees to quickly locate policies, SOPs, or project data. All internal documents are semantically chunked and stored with department-level metadata. Retrieval is based on vector similarity and keyword relevance.

8.5 Personalized Learning and Training Platforms

EdTech platforms use LLMs to create bite-sized learning modules and answer user queries. Course content is chunked into lessons, quizzes, and summaries. Chunks are tagged with learning outcomes and difficulty levels.

9 Conclusion

Chunking strategies play a crucial role in optimizing the use of context windows in large language models. By breaking down long or complex text into smaller, manageable parts, chunking ensures that LLMs can process information more effectively without missing important details or context.

The choice of chunking method depends on the type of content, the goal of the application, and the specific requirements of the task at hand. Advanced techniques such as dynamic chunking, hierarchical chunking, and embedding-aware chunking can further enhance the performance of LLM applications.

As context windows continue to grow in size, the importance of effective chunking strategies remains. By carefully designing and implementing chunking methods, developers can maximize the potential of LLMs and create more powerful and efficient applications.

10 References

1. Devlin, J., Chang, M.W., Lee, K., Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL.

2. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I. (2017). Attention Is All You Need. NeurIPS.

3. Blei, D.M., Ng, A.Y., Jordan, M.I. (2003). Latent Dirichlet Allocation. Journal of Machine Learning Research.

4. Lee, D.D., Seung, H.S. (1999). Learning the parts of objects by non-negative matrix factorization. Nature.

5. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language Models are Few-Shot Learners. NeurIPS.