Comparing Chunking Strategies and Embedding Models

This report provides a theoretical comparison of different text chunking strategies and embedding models, focusing on their impact on retrieval accuracy and latency in a vector database system. An empirical study would require extensive experimentation with diverse datasets and query types, which is beyond the scope of this project.

Chunking Strategies

Chunking is the process of breaking down large documents into smaller, manageable pieces (chunks) before embedding them. The choice of chunking strategy significantly impacts how well information can be retrieved.

Recursive Character Text Splitter:

Mechanism: This method attempts to split text using a list of separators recursively trying smaller separators if a chunk exceeds the maximum size. It aims to keep semantically related pieces of text together by splitting on logical boundaries first.

Retrieval Accuracy: Generally good. By prioritizing splits on natural breaks (paragraphs, sentences), it often preserves the context within a chunk, making it more likely that a relevant query will match a coherent piece of information. The overlap helps in capturing context that might span across two chunks.

Latency (Embedding & Retrieval):

Embedding: Moderate. Chunks are typically of a consistent, moderate size, leading to predictable embedding times.

Retrieval: Moderate. The number of chunks is optimized, leading to efficient vector searches.

Pros: Balances context preservation with chunk size, widely used and effective.

Cons: Can still split semantically related but syntactically separated information.

Sentence-based Chunking (or Semantic Chunking):

Mechanism: This method ensures that no sentence is ever split across chunks. Chunks are formed by aggregating complete sentences until a maximum chunk size is reached. While semantic chunking would involve more sophisticated techniques like identifying topic shifts or using sentence embedding to group highly similar sentences, a sentence-based approach is a good practical step.

Retrieval Accuracy: Potentially high for queries that align well with sentence boundaries or require complete sentences for context. It avoids the issue of a query matching only half a sentence.

Latency (Embedding & Retrieval):

Embedding: Can be variable. If sentences are very long, chunks might be larger, or if sentences are very short, many small chunks could be generated, impacting embedding batching efficiency.

Retrieval: Can be slightly higher if it results in a significantly larger number of smaller chunks compared to recursive, or lower if it results in fewer, well-defined larger chunks.

Pros: Guarantees contextual integrity at the sentence level; good for precise information retrieval.

Cons: Can create chunks that are too small (leading to many chunks and potentially diluted context) or too large (if a single sentence is very long).

Custom Fixed-Size Chunking:

Mechanism: Splits text into chunks of a predefined fixed character length, with a specified overlap between consecutive chunks. This is the simplest method.

Retrieval Accuracy: Variable, often lower than recursive or semantic. It might cut sentences or paragraphs arbitrarily, leading to incomplete contextual information within a chunk. The overlap helps mitigate this but doesn't solve the fundamental issue of semantic coherence.

Latency (Embedding & Retrieval):

Embedding: Fast and predictable. Chunks are uniform in size, making batch processing efficient.

Retrieval: Fast and predictable. The number of chunks is directly proportional to the document size and chunk size, leading to consistent search performance.

Pros: Simple to implement, predictable chunk sizes and counts.

Cons: Poor context preservation, can lead to lower retrieval accuracy if information is split awkwardly.

Embedding Models

Embedding models convert text into embedding that capture semantic meaning. The quality of whole directly influences retrieval accuracy.

all-MiniLM-L6-v2:

A small, efficient Sentence Transformer model.

Retrieval Accuracy: Good for its size. It performs well on a variety of semantic similarity tasks. For general-purpose document search, it provides a strong baseline.

Latency (Embedding & Retrieval):

Embedding: Very fast due to its small size (6 layers, 384 dimensions). Ideal for applications requiring low latency and efficient resource usage.

Retrieval: Fast. The 384-dimensional vectors are efficient for vector database search operations.

Pros: Excellent balance of speed, size, and performance; suitable for many real-world applications.

Cons: May not capture the most nuanced semantic relationships compared to much larger, more computationally expensive models.

General Considerations for Other Embedding Models (e.g. large Sentence Transformers)

Larger or more sophisticated models generally offer higher retrieval accuracy due to their ability to capture more complex semantic relationships and finer-grained distinctions.

Latency (Embedding): Larger models typically have higher embedding latency due to more parameters and higher dimensionality (e.g., 768, 1024, or even 1536 dimensions). This can be a bottleneck for real-time applications or large-scale ingestion.

Latency (Retrieval): Higher-dimensional embedding can slightly increase retrieval latency in vector databases, as more computations are required for distance calculations. However, modern vector databases are highly optimized for this.

Overall Findings and Recommendations

Retrieval Accuracy:

Chunking: Recursive and sentence-based chunking generally lead to higher retrieval accuracy because they preserve semantic context better within chunks. Fixed-size chunking is less reliable.

Embedding: Higher-quality, often larger, embedding models yield better retrieval accuracy. However, all-MiniLM-L6-v2 offers a very good trade-off for many use cases.

Latency:

Chunking: Fixed-size chunking offers the most predictable and often lowest embedding latency due to uniform chunk sizes. Recursive and semantic chunking might have slightly more variable latency but are still efficient.

Embedding: Smaller embedding models (like all-MiniLM-L6-v2) offer significantly lower embedding latency, which is crucial for high-throughput systems. Retrieval latency is also generally lower with lower-dimensional embedding.

Recommendations:

Start with Recursive Character Text Splitter: This is often the best default, offering a good balance between context preservation and performance.

Experiment with Sentence-based/Semantic Chunking: If high precision on sentence-level information is critical, or if documents have distinct semantic units, this approach can be superior.

Choose Embedding Model Based on Budget and Performance Needs:

For fast, cost-effective, and good-enough performance: all-MiniLM-L6-v2 is an excellent choice.

For maximum accuracy where latency/cost is less of a concern: Explore larger Sentence Transformers (e.g., all-mpnet-base-v2, bge-large-en-v1.5) or commercial APIs.

Monitor and Iterate: The optimal combination of chunking and embedding depends heavily on the specific dataset, query patterns, and application requirements. Continuous monitoring of retrieval metrics (e.g., precision, recall) and latency is crucial for fine-tuning the system.

This FastAPI application provides a solid foundation for building a robust document processing and retrieval system, allowing for experimentation with these critical components.