ChatGPT

Approaches to Multi-Page Document Transcription for TTS with LLM

Page-by-Page Processing

This simple method feeds one page at a time to the LLM with no explicit carry-over of context. Each page is transcribed independently. While easy to implement (just split by pages), it provides **very low context retention** – key information may be split across pages and continuity is lost. In fact, splitting text without regard to its structure can omit important details [1]. **Scalability** is high (pages can be processed in parallel with minimal overhead). **Implementation complexity** is low (just iterate pages). However, **podcast-style narration** suffers: transitions will likely sound abrupt and disjoint, making the output feel choppy.

- **Context Retention:** Very low (no cross-page context) [1] [2].
- **Scalability:** High (pages are independent; easy to parallelize).
- **Complexity:** Low (straightforward splitting by page).
- **Podcast Narration:** Poor – abrupt jumps and missing continuity make narrative flow weak.

Fixed-Size Overlapping Windows

This approach breaks the document into equal-sized chunks with overlaps (e.g. 2-page chunks with 1-page overlap) so each chunk shares content with the next. Overlaps help **preserve semantic continuity**, as each chunk contains a bit of its neighbor [3] [4]. **Context retention** is therefore moderate: the overlap ensures some context carries over. **Scalability** is moderate; chunking and overlap incur extra token usage (redundancy) and each chunk still needs an LLM call. **Complexity** is moderate: implementing sliding-window chunking (or using a library like LangChain's splitter) is straightforward but requires careful tuning of sizes/overlap. For **podcast narration**, this yields better flow than page-by-page: overlapping content provides natural "bridges" at chunk edges, though some repetition may occur.

- **Context Retention:** Moderate (overlapping content retains context between chunks [3] [4]).
- **Scalability:** Medium (adds overhead from overlapping tokens; still parallelizable).
- **Complexity:** Medium (requires sliding-window logic or text-splitting libraries).
- **Podcast Narration:** Fair – smoother than page-by-page due to overlap, but careful overlap design is needed to avoid repetition.

Logical-Boundary Chunking

This strategy splits text at natural boundaries (headings, sections, paragraphs) so each chunk is a semantically complete unit. By aligning with the document's structure, chunks tend to be self-contained, yielding **high context retention** within each piece. For example, semantic or "embedding-based" chunking groups related information together so that chunks "retain full meaning" [5]. **Scalability** is moderate: fewer, larger chunks may be produced, but splitting requires content analysis (e.g. detecting headings or topics). **Implementation complexity** is higher than fixed splitting: it may involve NLP tools or heuristics to detect logical breaks. In terms of **podcast-style narration**, this is effective: each chunk ends at a logical point, so the read-aloud voice flows more naturally.

- **Context Retention:** High (chunks align with complete thoughts/sections; meaning preserved [5]).
- **Scalability:** Medium (chunk sizes vary; need to fit model context limits).
- **Complexity:** Medium–High (requires parsing or NLP to detect boundaries, possibly custom rules).
- **Podcast Narration:** Good – coherent sections flow naturally in speech, with fewer jarring cuts.

Memory-Augmented Processing

In this approach, the LLM is given a running "memory" or summary of prior chunks when processing each new part. For example, a *refine-chain* technique passes every chunk along with a summary of previous chunks [6] . This yields **very high context retention**, since each prompt is informed by earlier content. **Scalability** is low: the process is sequential (you must finish one chunk and update the summary before moving on) and cannot be easily parallelized. **Implementation complexity** is high: it requires generating and managing intermediate summaries or context representations (often via extra LLM calls). For **podcast narration**, memory-augmentation gives the best continuity – the narration can reference earlier points and maintain tone consistently. However, it is more expensive and slower.

- **Context Retention:** Very high (ongoing summaries ensure full continuity [6] [7]).
- **Scalability:** Low (sequential processing; multiple passes).
- **Complexity:** High (requires managing context/summaries or a memory module).
- **Podcast Narration:** Excellent – produces the most cohesive, story-like output with smooth transitions.

Hybrid and Best Practices

Hybrid methods combine the above ideas. For example, one might split by logical boundaries *and* use slight overlap or a rolling summary to bridge gaps. Another best practice is iterative summarization: process each section (by heading) and update a short summary to carry context forward, blending semantic chunking with memory [8] . Empirical guidance suggests that some overlap or context caching is important: "keeping some overlap ensures semantic context doesn't get lost" [4] [3] . In practice, a mix of techniques often works best – e.g. chunk at paragraphs, overlap a sentence or two between chunks, and append a brief recap of prior points to each prompt. This balances coherence and cost.

- **Hybrid Strategies:** Combine structure-based splits with overlap or summary-contexting; e.g. split at headings, overlap buffer sentences, and feed forward key points (a "refine" style) to maintain flow [4] [8] .

Comparison of Methods

Approach	Context Retention	Scalability	Complexity	Podcast Narration
Page-by-Page	Very Low	High (easy to parallelize)	Low	Poor (choppy, disjoint)

Approach	Context Retention	Scalability	Complexity	Podcast Narration
Overlapping Windows	Moderate (improved by overlap)	Medium (some redundancy)	Medium	Fair (better continuity, some repetition)
Logical Boundaries	High (complete thoughts)	Medium (chunk sizing)	Medium–High	Good (coherent segments)
Memory-Augmented	Very High (full context)	Low (sequential)	High	Excellent (smooth, cohesive)

Each approach offers trade-offs. Page-by-page is simplest but loses cross-page flow; overlapping chunks improve context at modest extra cost [3] [4]. Chunking on logical breaks preserves meaning best [5] but requires smarter splitting. Memory-based methods achieve the most natural, narrative output [6] [7] but are resource-intensive. In practice, combining these methods (e.g. structural chunking with some overlap and contextual summaries) often yields the best balance of continuity and efficiency.

Sources: Prior work on long-text LLM processing highlights these trade-offs. For instance, naive fixed chunks can omit key info [1], while overlapping or semantic splits preserve context [3] [5]. Sequential "refine" or memory chains retain coherence [6] [7] but incur more computation. These insights guide best practices in preparing multi-page content for TTS narration.

---

[1] [6] **Summarize Podcast Transcripts and Long Texts Better with NLP and AI | by Isaac Tham | TDS Archive | Medium**
https://medium.com/data-science/summarize-podcast-transcripts-and-long-texts-better-with-nlp-and-ai-e04c89d3b2cb

[2] [3] **Understanding Chunking Algorithms and Overlapping Techniques in Natural Language Processing | by Jagadeesan Ganesh | Medium**
https://medium.com/@jagadeesan.ganesh/understanding-chunking-algorithms-and-overlapping-techniques-in-natural-language-processing-df7b2c7183b2

[4] **Chunking Strategies for LLM Applications | Pinecone**
https://www.pinecone.io/learn/chunking-strategies/

[5] [8] **LLM Chunks — Breaking Down Context Efficiently | by Sushant Gaurav | Medium**
https://sushantgaurav57.medium.com/llm-chunks-breaking-down-context-efficiently-236dafe7b564

[7] **Recursively summarizing enables long-term dialog memory in LLMs | by mike | May, 2025 | Artificial Intelligence in Plain English**
https://medium.com/ai-in-plain-english/recursively-summarizing-enables-long-term-dialog-memory-in-llms-6b0c6fbd1bcb