ChatGPT

RAGシステムにおける技術文書のテキストチャンク戦略

Retrieval-Augmented Generation（RAG）システムでは、大規模言語モデル(LLM)に外部知識を与えるため、**文書をチャンク（分割）してベクトルデータベースに格納**します [1]。適切なチャンク戦略を採用することで、**検索精度**や**回答の正確さ**が向上し、逆に不適切なチャンクは文脈喪失やノイズを引き起こします [2] [3]。以下では、技術文書（論文・報告書・発表スライドなど）におけるチャンク手法について、文書タイプ別の最適戦略、コスト考慮、LLM活用例、一般的なチャンクサイズ、そして検索精度・回答精度への影響の観点から整理します。

文書タイプ別の最適チャンク戦略

技術文書の種類ごとに、構造や内容に適したチャンク分割アプローチがあります。以下の表に、文書タイプ別の推奨チャンク戦略とその理由をまとめます。

文書タイプ	推奨チャンク手法	理由・補足
研究論文（Academic Paper）	セマンティックチャンク or セクション単位の分割	論文は内容が高密度で複雑なため、意味やトピックのまとまりごとに分割するのが有効です。例えば**章や節（「方法」「結果」など）ごとに区切る**ことで論理的な塊を保てます [4]。また各文をベクトル化して類似性で区切るセマンティック手法では、**トピックが変わる箇所でチャンク境界**を検出し、意味的に一貫した塊を作れます [5] [6]。論文は段落だけでは話題の切れ目が明確でない場合も多いため、このような**意味ベースの分割**が適しています [7]。
技術報告書・マニュアル	構造ベースのチャンク（見出しや段落単位）＋必要に応じ固定長併用	技術報告書は一般に章立てや箇条書きなど**一定の構造**があります。MarkdownやPDFの**見出しレベル（例：`##`）で区切る**と論理的まとまりを維持できます [8]。段落単位のチャンクも「1段落=1チャンク」で一つの話題を含むため有効です [9]。文章構造が明確な場合はこの**コンテントアウェア（内容構造に基づく）**な分割で文脈が保たれます [8]。一方、ログファイルやコードの説明のようにフォーマットが均一なテキストでは、**固定長チャンク**（例：500単語または2000文字ごと）の単純分割でも機能します [10]。報告書ではまず見出し・セクションごとに分割し、それでもチャンクが長すぎる部分はさらに固定長でスプリットする、といった**ハイブリッド戦略**も有効です [4] [11]。

文書タイプ	推奨チャンク手法	理由・補足
発表スライド（プレゼン資料）	スライド単位のチャンク（ページ区切り）＋箇条書きの統合	スライド資料は1枚ごとに話題が完結することが多いため、**各スライドを1チャンク**とするのが自然です。実際、ドキュメントをページ単位で区切る手法では、異なるページの内容が同一チャンクに混在しないようにし、**新しいページ（スライド）が現れたら新チャンクを開始**します [12]。こうすることでスライドごとの文脈が保たれます。スライド内のテキスト量が非常に多い場合（長い文章や段落がある場合）は、そのスライド内で段落ごとに分割するか、**箇条書きリストなら複数項目をまとめて1チャンク**にするなど調整します。一般にスライドはテキストが少ないため、**スライド1枚＝チャンク1つ**で十分であり、過度に細かく分割しすぎないことがポイントです。

補足: 上記以外にも、長大な技術文書（本や長大なマニュアル）には**階層的チャンク**を用いる例があります。例えば**教科書**では、まず章全体を要約した大きなチャンクを作り、その下位に章内の節ごと・段落ごとの小チャンクを用意する、といった**マルチレベルの階層チャンク**戦略です [13] [14]。またプログラムのソースコードやログなど**特殊なフォーマット**では、関数やクラス単位、ログエントリ単位などドメイン固有の粒度でチャンク化するケースもあります [15]（このようなモダリティ別チャンク戦略を**Modality-Specific Chunking**と呼び、テキスト・表・画像など各要素を別々に処理します [16] [17]）。

コスト観点からのチャンク戦略

チャンク戦略は**コストや処理時間**の観点でも選択肢が分かれます。高度なチャンク処理ほど計算資源やAPI利用料金が増大するため、文書の重要度や分量に応じて使い分けるのが現実的です。

- **ルールベースの分割（低コスト）:** 固定長チャンクや構造に基づくチャンクは、シンプルなアルゴリズムで高速に処理できます。大量の文書を扱う場合や**コストを抑えたい場合**、まずはこれら**安価な手法**で一括処理するのが合理的です [10] [8]。例えば数百ページに及ぶ技術資料群をすべて高度なモデルで解析するとコストが膨大になるため、まずは**段落・見出し単位**で機械的に分割する、といった方針です。

- **セマンティック・Embedding利用（中コスト）:** 文をベクトル化して類似度クラスタリングするセマンティックチャンクは、LLMより軽量ですが**通常の分割よりは計算コスト**がかかります [4]。とはいえ、すでにベクトルデータベースに格納するため**文書全体をEmbeddingする工程は不可欠**なので、その延長で適用できる手法です。意味的な境界を検出する計算（類似度計算）やトピック変化の検知を行う分だけの**追加コスト**と考えられます。大規模データでは完全なLLMよりは安価なため、**中規模の重要文書**にはこのアプローチが現実的です。

- **LLMを用いたチャンク（高コスト）:** 大規模言語モデル自身に文書を読ませてチャンク分割や要約を行う方法は**最も強力だがコストが高く**、処理も遅くなります [18] [19]。そのため**「重要度が高い一部の文書だけLLMで精緻にチャンクし、その他多数は自動ルールで安価に処理」**という使い分けが有効です。例えば社内の基幹技術文書や法的文書など**精度が特に重要な文書**にはGPT-4など高性能モデルでセクション要約やキーポイント抽出を行いチャンク化する一方、FAQ集やログなど**多少粗くても許容できる文書**は機械的に段落分割する、といったハイブリッド運用です。LLMチャンクは**品質最優先・コスト度外視**の場合に適し（法律契約書、学術論文など） [18]、その際でもモデルへのプロンプト回数を減らす工夫（1回のプロンプトで複数チャンク生成など）でコスト抑制を検討します。

- **チャンク処理のタイミング:** コスト最適化のため、**事前チャンク**と**オンデマンド（事後）チャンク**の選択もあります。通常は全ドキュメントを前もってチャンク・Embeddingしますが、利用されない文

書まで変換するのは無駄になる可能性があります [20]。そこで**Post-Chunking**と呼ばれる戦略では、まず文書全体をEmbeddingだけしておき（粗粒度のインデックス作成）、ユーザ質問に関連してヒットした文書のみをその場で細かくチャンク化・再埋め込みする方法もあります [21]。これにより**必要な文書にだけ詳細なチャンク処理**を行うため、最初からすべてを細分しストレージを消費するより**コスト効率を高める**ことが可能です。ただし初回クエリ時に遅延が増える、実装が複雑になるといったトレードオフがあります [20]。

LLMを使ったチャンク処理の具体例

LLM（大規模言語モデル）をチャンク生成に活用することで、人間が行うような**意味単位での柔軟なテキスト分割**が可能になります。具体的なアプローチ例をいくつか挙げます。

- **意味的まとまりの自動抽出:**　LLMに文書を読ませて「内容が切り替わる箇所で区切りを入れてください」と指示し、**論理的な境界でテキストを分割**させる方法です。モデルは文脈を理解しているため、単純な文字数ベースのカットではなく**話題の転換点**でチャンクを提案できます。例えば「以下の論文を読み、各段落がどのテーマについて書かれているかに基づいてセクションを分割してください」とプロンプトし、モデルから返ってきた区切り位置でテキストを切るような使い方です。これは一種の**LLMによるセマンティックチャンク**であり、Embeddingの類似度では捉えにくい**高レベルな意味のつながり**を考慮できます。

- **要約付きチャンク（文脈エンリッチメント）:**　チャンク単体では文脈が不足しがちな場合、LLMに**各チャンクの要約や見出しを生成させ**てそれをチャンクに付加する手法があります。Anthropic社は**Contextual Retrieval**という手法を提案しており、Claudeに文書全体と対象チャンクを与えて「チャンクの内容を要約・説明する一文」を生成、それをチャンク先頭に付け加えてEmbeddingすることで**そのチャンクが何についての記述かを明示**しました [22] [23]。例えば「このチャンクはACME社の2023年Q2業績に関するSEC報告書の一部である…」といった説明文を加えることで、チャンク単体にもドキュメント全体の文脈が反映され、**検索時に適切な情報を拾いやすく**なります [23]。このように**チャンク内容＋その要約**をセットにEmbeddingすれば、細切れにした情報にも高次の意味を持たせられます（文献では「Context-Enriched Chunking」として紹介 [24]）。

- **LLMによる要約分割 (Summarize then Split):** 長大な文書をそのまま細切れにすると文脈断裂が起きる場合、LLMで**要約してから分割**するアプローチがあります。例えば100ページの技術報告書なら、まずLLMに各章の要約を作らせ、その要約文をチャンクとして扱うか、要約を元に再度詳細部分を切り出す方法です。これにより、情報量を圧縮しつつ**重要点を含むチャンク**を得ることができます。要約自体もEmbeddingして検索に用いることで、詳細チャンクでは拾えない高レベルの回答（「この章の概要は？」など）にも対応できます。この種の手法は**階層的チャンク＋要約**とも言え、上位レベルチャンクにLLM要約を活用するイメージです [13] [14]。

- **Propositionベース分割:**　LLMの能力を使い、**文章を論旨ごとの命題に分解**する手法です [25]。例えば「この段落に含まれる主張を箇条書きにしてください」とLLMに頼み、その箇条書き一つひとつを独立チャンクと見なすアプローチです。LLMは人間のように文を再構成できるため、単純な文分割よりも**論理的な完結性**を持ったチャンクになります [25]。またモデルに**キーポイントをハイライト**させて抽出することで、重要な情報だけを含むチャンクを生成することも可能です [25]。このようなLLMベースの手法は特に**重要文書（契約書、研究論文など）で高い効果**を発揮しますが、前述の通りAPIコストと実行時間の負担も大きくなります [26]。

- **エージェント型チャンク (Agentic Chunking):** さらに高度な例として、**AIエージェントに文書構造を分析させ最適なチャンク戦略を文書ごとに選択させる**方法があります [27]。エージェントはまず文書全体を見て、例えば「この文書はMarkdown形式だから見出しで区切ろう。この部分は密度が濃いから命題ベースで、更にコードブロックは別で…」というように、**複数のチャンク手法を組み合わせて**

適用します [27]。実装上はLLMに「この文書に最も適したテキスト分割方法を考えて実行してください」といったプロンプトを与える形です。Agentic Chunkingは極めて柔軟で高品質な結果が期待できますが、文書解析から分割まで**強力なモデルへの多数回の問い合わせ**が必要となるためコスト・速度面では最も贅沢な手法になります [19]。従って**品質が最優先でコストに糸目を付けないケース**（企業の基幹ナレッジベース構築など）以外ではあまり現実的ではありませんが、将来的な自動化の方向性として注目されています。

一般的なチャンクサイズの知見

チャンクサイズ（1チャンクあたりのテキスト量）は、チャンク戦略で最初に決めるべき重要パラメータです。チャンクが**大きすぎる**とEmbeddingモデルのコンテキスト上限を超えて切り捨てられたり、ベクトルがトピック混在で曖昧になります [28] [29]。逆に**小さすぎる**と文脈が細切れになりすぎ、人間・モデル双方が意味を取りにくくなります [30]。適切なサイズを決めるための一般的な知見を以下にまとめます。

- **トークン数基準:** Embeddingモデルの最大長に収まる範囲で、**数百トークン程度**を上限にします。Anthropicは「通常、1チャンクはせいぜい数百トークン程度にする」と述べています [31]。具体的な目安として**200〜500トークン**程度がよく使われる範囲です（モデルの性能と文書内容によります）。例えばOpenAIのtext-embedding-ada-002は約8000トークンまでEmbedding可能ですが、それに近い長さを丸ごと1チャンクにすると「長すぎて中間の情報を見落とす」問題があるため [32]、実際にはもっと短く区切ります。Pinecone社も、**128や256トークンの小さなチャンクから、512・1024トークンの大きめチャンクまで試し**て適切な長さを見極めることを推奨しています [33] [34]。

- **文字数・文数基準:** 日本語などでは文字数や文の個数でチャンク長を考えることもあります。おおよそ**250トークン ≒ 1000文字** と言われ [35]、500トークンなら2000文字程度が目安です。文数で言えば内容にもよりますが、**3〜5文程度**をまとめて1チャンクにする例が多く見られます。段落単位では長短ありますが**平均的な段落1つ**がちょうどチャンク1つに相当する場合も多いです [9]。いずれにせよトークン数ベースでチェックするのが正確ですが、感覚的には「数文から長くても数十文」が1チャンクになるイメージです。

- **オーバーラップ:** チャンク分割時には、**隣接チャンク間に一部テキストの重複**（Overlap）を持たせるのが一般的です。典型的には**チャンク長の10〜20%**を前のチャンクから重ねて含めます [36]。これにより**チャンク境界付近の文脈切れ**を防ぐことができます [37]。例えば300トークンずつのチャンクで20%オーバーラップなら、チャンクあたり約60トークンを次にも重複して含めます。重複率を上げすぎるとデータ冗長が増えますが、10%前後のオーバーラップは**重要情報がチャンク境界で途切れないための保険**として有効です [38]。

- **チャンクサイズ選定と実験:** 最適なチャンクサイズはコーパス（文書群）の特性やユーザクエリの傾向にも依存するため、**複数サイズで試行評価**するのが望ましいとされています [39] [40]。例えば128, 256, 512トークンなど異なる長さでEmbeddingインデックスを作り、いくつかの質問に対する検索結果の精度（ヒット率、再現率、回答の関連性など）を比較する方法です [39] [41]。この実験により、例えば「小さすぎると回答が断片的になる」「大きすぎると関係ない内容まで引っかかる」といった傾向が可視化できます。一般には**チャンクが自立して意味を成すか**が重要な基準であり、「そのチャンクだけ読んで人間が内容を理解できるか」を目安にサイズを調整すると良いと言われます [42] [43]。実際「周囲の文脈なしでも意味が通る塊は、検索にも有用」だからです [43]。

チャンク戦略が検索精度・回答精度に与える影響

チャンクの切り方は**ベクトル検索の命中精度**および**最終的な回答の正確性**に大きな影響を与えます [44]。適切なサイズ・粒度でチャンク化すると、必要な情報を漏れなくかつノイズ少なく取得でき、LLMが根拠に基づいた回答を生成しやすくなります。以下、チャンク戦略の違いによる精度面の影響を整理します。

- **小さなチャンクの利点と欠点:**　チャンクを細かく分割すると、**各チャンクが一つの明確なトピックや事実を含む**ためベクトル検索でのマッチ精度が上がります [45] [46]。embedされたベクトルが特定の内容を精密に表すため、ユーザクエリとの類似度比較で**ピンポイントに該当箇所をヒット**させやすくなります [45]。Harshadによる実験でも、**極小チャンク（例：25トークン）では質問に対するコンテキスト精度（Precision）が向上**したと報告されています [47]。これは例えば「technetiumの応用」に関する問いに対し、小チャンクだと**「医療イメージングへの応用」という一文だけを含むチャンク**が返り精度が高い、という具合です [47]。さらにモデルに渡すコンテキストが過度に長くならないため、**幻覚（ハルシネーション）の抑制**にも役立ちます。モデルは**与えられた事実ベースの短い文脈**から回答を作るため、余計な想像をしにくくなるためです [48]。一方でチャンクが小さすぎると**文脈が断片化**し、単体では意味や指示対象が不明確になることがあります [30]。人間が一文だけ読んでも「誰が何をした話か不明」であるように、モデルも扱いにくく、**必要な関連情報が別のチャンクに分散**すると回答生成に十分な材料が得られない恐れがあります [49]。このため小チャンクでは**検索側の再現率（Recall）**が低下しがちです [50]。先の実験では25トークンチャンクは、質問に関連する情報を**全てカバーする割合（Context　Recall）が低下**した例が示されています [50]。総じて**「精度向上と引き換えにカバレッジ低下」**が小チャンクの特徴です。

- **大きなチャンクの利点と欠点:** チャンクを大きくすると、一つのチャンクに**より多くの文脈や関連情報**を含められるため、ユーザの質問に対し**包括的な検索ヒット**を得やすくなります [51]。例えば「technetiumの将来研究課題は？」という問いに対し、1024トークンの大きなチャンクなら**応用と課題の両方を網羅した段落**が丸ごと引き当てられるため、**文脈の再現率（Recall）が向上】します** [51]**。また一度にモデルに渡せる情報量が多いので、回答生成時にも**広い文脈を参照して包括的な回答**ができる利点があります** [50]**。しかしデメリットとして、**チャンク内に無関係な内容が混在**しやすくなります。Embed時には複数トピックが平均化されたベクトルになってしまい、特定クエリとの類似度が薄まり**検索漏れ**を起こす可能性があります** [52]**。実際、「チャンクが大きすぎると複数の話題が混ざり、どれも中途半端なベクトル表現になる」と指摘されています** [52]**。このため**検索精度（Precision）は低下**しやすく、ノイズの多いチャンクが返ってくることもあります** [47]**。モデル応答の面でも、長大なコンテキストは**Attentionの拡散による精度劣化**や「途中の情報を見落とす現象（Lost in the Middle）」を招きます** [53]**。LLMは長い入力では冒頭と末尾に注意が偏り、中間の詳細を誤ることが知られているためです** [53]**。要するに**「包括性向上と引き換えに精度低下」**が大きなチャンクの特徴です。

- **チャンク戦略と最終回答精度:** 上記の通り、小チャンクと大チャンクは一長一短であり、中間的なサイズや工夫次第でバランスを取ることが可能です。Harshadの実験では**中程度のチャンク（例：200トークン前後）**が**PrecisionとRecallのバランスが良好**で、質問に対する関連度スコア（Context Relevancy）が高くなったとされています [54]。具体例として、200トークン程度なら「technetiumの応用」に関する2〜3文と「将来課題」に関する記述を一つのチャンクで含み、質問に対し両面から関連情報を提供できた、という分析です [54]。また検索段階では小チャンクで精度重視し、応答生成時に周辺文脈を追加する**ハイブリッド手法**も提案されています [55] [56]。Sentence Window Retrieval（文単位検索＋窓付き文脈拡張）では、まず**検索は1文チャンクで行い**（精度重視）、ヒットした文の前後数文を取り出してモデルに渡すことで**コンテキスト欠如を補う**という工夫が紹介されています [55] [57]。このようなアプローチにより**精度と文脈の両立**が図れる可能性があります。

- **回答への影響まとめ:** 良いチャンク戦略により、RAGシステムの**回答の正確さ・一貫性**が向上します。チャンクが**話題ごとに適切に分割**されていれば、検索エンジンは質問に対応する事実を見逃しにくく

なり（Recall向上）、無関係な部分を誤って拾うことも減ります（Precision向上） [58] 。その結果、LLMは**関連する根拠のみ**をもとに解答を作れるため、的確で裏付けのある回答が得られます。逆に不適切なチャンク化（文脈を無視した切断や極端なサイズ不均一など）は、**検索ミスや不要情報の混入**を招き、最終回答の誤りや不明瞭さにつながります [3] 。したがってRAGにおいては、**チャンキングこそが性能の土台**と言われます [44] 。実運用では文書タイプに応じた最適戦略を選びつつ、上記のようなトレードオフを踏まえてチャンクサイズ・手法を調整することが重要です。適切にチューニングされたチャンク戦略は、**必要十分な文脈を持つ検索単位**を提供し、結果的に高い検索精度と信頼できる回答生成に寄与します [59] [60] 。

---

[1] The Secret to Efficient RAG: A Step-by-Step Guide to Chunking and Counting Your Vectors - DEV Community
https://dev.to/aairom/the-secret-to-efficient-rag-a-step-by-step-guide-to-chunking-and-counting-your-vectors-25go

[2] [3] [9] [16] [17] [24] 11 Chunking Strategies for RAG — Simplified & Visualized | by Mastering LLM (Large Language Model) | Medium
https://masteringllm.medium.com/11-chunking-strategies-for-rag-simplified-visualized-df0dbec8e373

[4] [8] [10] [11] What chunking strategies work best for document indexing?
https://milvus.io/ai-quick-reference/what-chunking-strategies-work-best-for-document-indexing

[5] [6] [7] [13] [14] [15] [18] [19] [20] [21] [25] [26] [27] [30] [36] [37] [38] [41] [42] [44] [45] [46] [48] [52] [53] [59] [60] Chunking Strategies to Improve Your RAG Performance | Weaviate
https://weaviate.io/blog/chunking-strategies-for-rag

[12] [28] [29] [35] Chunking for RAG: best practices | Unstructured
https://unstructured.io/blog/chunking-for-rag-best-practices

[22] [32] [33] [34] [39] [40] [43] Chunking Strategies for LLM Applications | Pinecone
https://www.pinecone.io/learn/chunking-strategies/

[23] [31] [49] Contextual Retrieval in AI Systems \ Anthropic
https://www.anthropic.com/engineering/contextual-retrieval

[47] [50] [51] [54] [55] [56] [57] [58] Evaluating the Optimal Document Chunk Size for a RAG Application | by Harshad | Medium
https://harshadsuryawanshi.medium.com/evaluating-the-optimal-document-chunk-size-for-a-rag-application-9cb482365bbf