F-029

RAG として AIChatBot に与えるデータベースの構造に関する一考察
On Database Structures for Retrieval-Augmented Generation in AI Chatbots

川越 航太† 須藤 康裕† 山田 宏樹‡ 松本 一教† 一色 正男†
Kota Kawagoe Yasuhiro Sudo Hiroki Yamada Kazunori Matsumoto Masao Issiki

1. はじめに

大規模言語モデルが外部の知識ベースから必要な情報を検索し、その情報を基に回答を生成する過程において、ベクトルデータベース（Vector Database: VDB）での類似性を基に関連性の高い情報を見つけ出す仕組みを RAG(Retrieval-Augmented Generation)と呼ぶ。本稿では、一般的な社内文書としてよく用いられるドキュメント形式や PDF 形式のファイルを、フォーマット変換した後に VDB 化することでより高い精度での検索をねらう。

2. VDB 生成手法

実験に用いるデータは FAQ（Q&A のセット）であり、ドキュメント形式で提供されるものである。これを、PDF、TXT、XML、JSON、JSONL 形式に変換した後に VDB を構築する。VectorStoreIndex への登録前に、Python で実装した前処理スクリプトを用いて①テキスト抽出、②チャンク分割、③メタデータ付与、④埋め込み生成を行う。全スクリプト共通で埋め込みモデルには intfloat/multilingual-e5-large、LLM には Gemini 2.0-flash を採用する。チャンク長は 1,000 文字、オーバーラップを 200 文字とする。

3. フォーマット別前処理実装

各スクリプトの主要処理を表 1 に示す。ドキュメント形式では分割単位を段落とし、付与メタデータはファイル名および段落番号である。PDF 形式では分割単位をページごとにし、付与メタデータはファイル名およびページである。もっとも単純な TXT 形式では分割は行わず付与メタデータはファイル名のみとした。CSV 形式では分割単位をレコードとし、付与メタデータはファイル名と行である。XML 形式では分割単位を<エントリ番号>要素 とし付与メタデータはファイル名およびエントリ番号とした。JSON 形式では分割単位をオブジェクトとし、付与メタデータはファイル名およびエントリ番号とした。最後に JSONL 形式では分割単位を行とし、付与メタデータはファイル名およびエントリ番号とした。また図 1 にそれぞれのファイル形式のデータ構造例を示す。

4. 実験結果 1（100 問）

Q&A が 100 セットの FAQ をもとに VDB を作成し、100 セット中の無作為な質問文書を入力として回答を得る実験を行った。それぞれのファイル形式で作成した VDB による回答精度（正答率）を表 2 および図 2 に示す。

表 1: 使用ライブラリおよび分割単位

ファイル形式	抽出ライブラリ	分割単位	付与メタデータ
DOCX	python-docx	段落ごとにチャンクに分割	ファイル名＋段落番号
PDF	PyPDF2.PdfReader	ページごとにチャンクに分割	ファイル名＋段落番号
TXT	open().read()	全文をチャンクに分割	ファイル名
CSV	pandas.read_csv	行ごとにテキストをチャンクに分割	ファイル名＋行番号
XML	xml.etree.ElementTree	ルート直下の<エントリ番号>要素ごとにチャンクに分割	ファイル名＋エントリ番号
JSON	json.load	オブジェクトごとにチャンクに分割	ファイル名＋エントリ番号
JSONL	json.loads	行ごとにチャンクに分割	ファイル名＋行番号

表 2: 100 問での成功/失敗回数

ファイル形式	成功回数	失敗回数
DOCX	7	3
PDF	9	1
TXT	9	1
CSV	9	1
XML	9	1
JSON	10	0
JSONL	10	0

元データのドキュメント形式では 70%の正答率しか得られていないのに対し、他のすべてのファイル形式で正答率の向上が見られる。とくに JSON 形式と JSONL 形式ではすべてのケースで正答を検索することに成功している。

† 神奈川工科大学 Kanagawa Institute of Technology
‡ manaable 株式会社 manaable Co. Ltd.

図 1: ファイルの文書構造

5. 実験結果 2（1000 問）

1000 問の大規模クエリセットを用い、各ファイル形式の検索成功率をそれぞれ 10 回再評価した。結果を表 3 および図 3 に示す。この実験では、100 問での結果とは異なり、CSV が 8/10（80%）と相対的に最良の成績を示した。

表 3: 1000 問での成功/失敗回数

ファイル形式	成功回数	失敗回数
DOCX	4	6
PDF	6	4
TXT	6	4
CSV	8	2
XML	7	3
JSON	7	3
JSONL	7	3

6. 考察

100 問の実験では JSON と JSONL が完全成功率（10/10）を示し、CSV・PDF・TXT・XML も 90%（9/10）と高い水準を維持していた。しかし、Q&A 件数を 1000 に拡大すると JSON 系の成功率は 70%（7/10）に低下したのに対し、CSV は 80%（8/10）を維持して最良成績を保った。CSV のセル区切りという単純構造は、大規模化に伴う例外パターンを吸収しやすく、前処理ロジックが比較的堅牢に機能したためと考えられる。一方、DOCX はレイアウトやネスト依存性により成功率が 40%へ急落し、PyPDF2 を用いた PDF からのテキスト抽出も改行コードや空白・タブの不統一が文脈を断片化させやすかった。

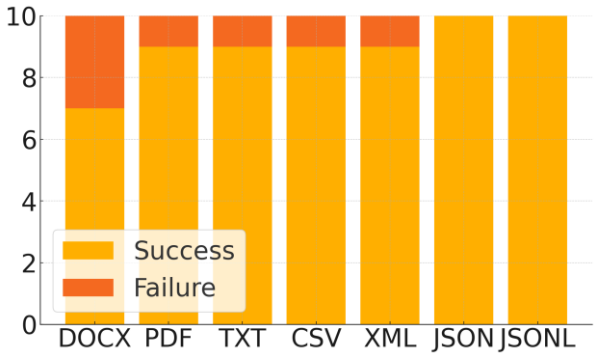

図 2: 100 問での成功/失敗回数

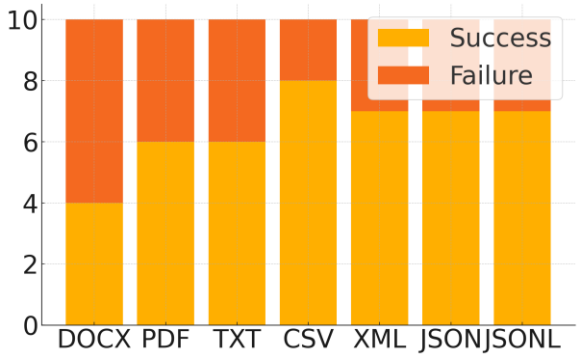

図 3: 1000 問での成功/失敗回数

また、チャンクの分割単位が小さすぎると情報不足、大きすぎるとノイズ混入を招くため、最適な粒度設定が検索性能に大きく影響することが示された。これらを踏まえると、大規模運用では構造化データ形式の利用と、改行・エスケープ処理を自動補正するクレンジングを組み合わせる二段構えが、検索性能と運用コストの両面で現実的かつ効果的なアプローチと言える。

7. おわりに

これらの知見は、実サービスで大規模 Q/A データを扱う際のフォーマット選択と前処理設計に直接的な指針を与える。特に、① 可能であれば CSV など構造が単純で堅牢な形式に統一する、② スキーマ自動検出や改行・エスケープ補正を自動化し、例外パターンを機械的に吸収する、③ PDF/DOCX については構造化データにしやすいような書き方にするという三点が重要である。今後は、前処理アルゴリズムの自動チューニング機構とエラー検出・フィードバックループを組み込み、安定した検索性能を維持できるベクトルデータベース基盤の構築を目指す。

参考文献

[1] 梶川 怜恩, 神田 峻介, 赤部 晃一, 小田 悠介,”ベクトル検索におけるテキスト構造化の効果分析”言語処理学会 第 31 回年次大会 (2025)

[2] Toni Taipalus, “Vector database management systems: Fundamental concepts, use-cases, and current challenges”, Cognitive Systems Research, Vol.85 (2024)