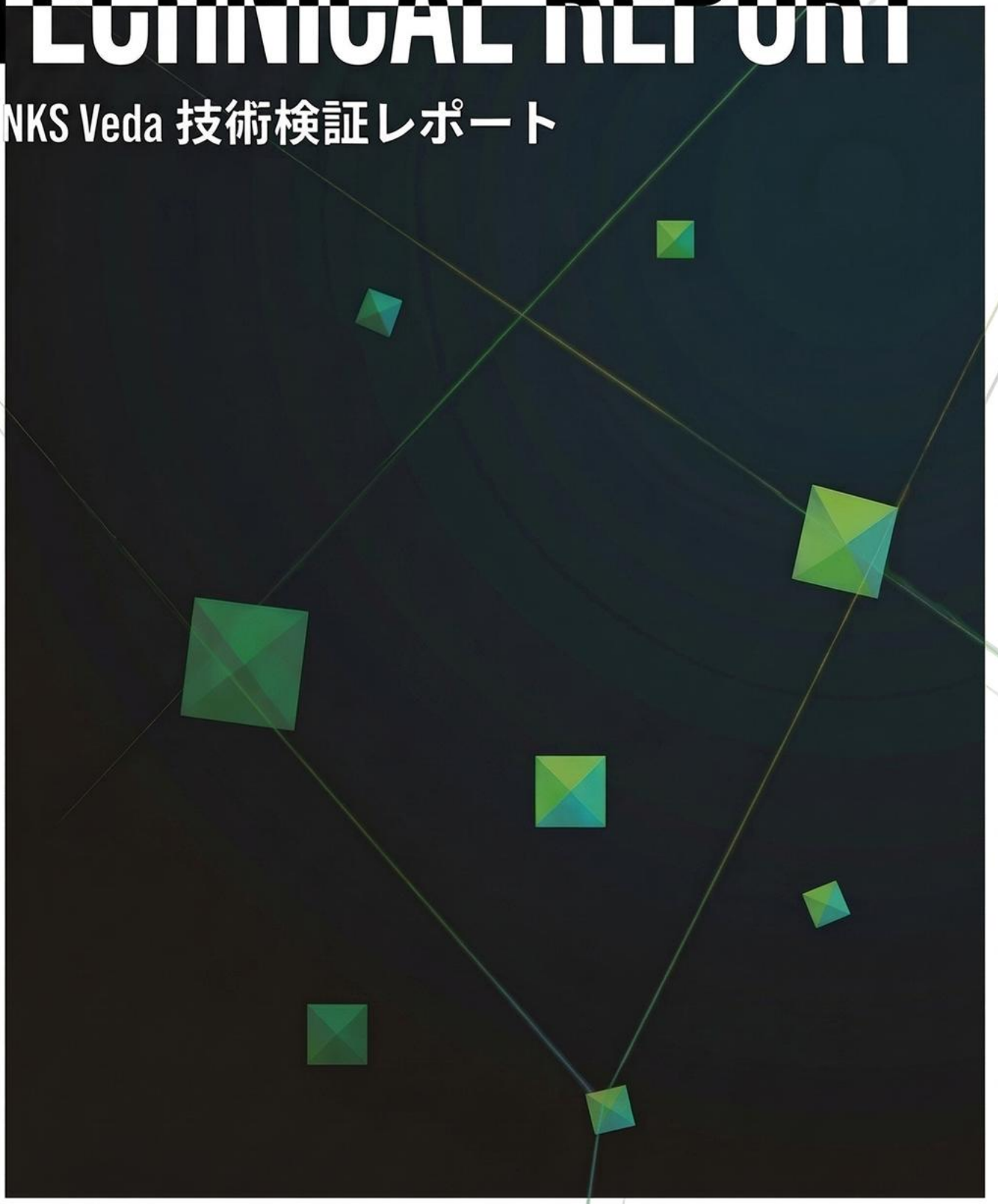
2025

LINKS Veda TECHNICAL REPORT

LINKS Veda 技術検証レポート



Project LINKS
by MLIT

第 0 章　はじめに

本章では、本ドキュメントの目的・記載範囲・全体構成を示す。

0.1　本ドキュメントの目的

本ドキュメントは、国土交通省が進める Project LINKS が 2025 年度に開発した実証環境であるデータ構築基盤システム「LINKS Veda(v2)」（以下、「Veda」と呼ぶ。）の技術検証レポートである。Veda の開発・実証を通じて、以下の 3 点を報告することを目的とする。

- どのような技術をなぜ選択したか。
- どのような課題に直面し、どのように解決したか。
- 実証の結果として何が有効で何が不十分だったか。

本ドキュメントは技術選定の判断理由（Why）と検討・検証のプロセス（Process）の記録を主眼に置いている。単に実装した機能を列挙するのではなく、なぜその設計・技術を採用したのかという判断の背景と、実際に試してみた結果の考察を中心に記述する。

想定読者は以下の通りである。

- 国土交通省の担当者：本実証の技術的成果と有用性の評価を確認したい方。
- 類似プロジェクトの関係者：行政データを活用するシステム構築の参考にしたい方。
- 一般開発者：大規模なデータ整備における技術選定のプロセスを学びたい方。

本ドキュメントでは、読者が以下の水準の知識を有していることを前提とする。

- 既知として扱う知識：PC や Web ブラウザの基本操作、クラウドサービス（AWS・Google Cloud 等）の概念的な理解、データベースや API といった IT の基礎用語。
- 説明を付して扱う知識：LLM（大規模言語モデル）・OCR（光学文字認識）・VLM（視覚言語モデル）等の AI 技術、GIS（地理情報システム）、データエンジニアリング（メダリオンアーキテクチャ・Parquet 等）、Function calling・プロンプトエンジニアリング等の AI アプリケーション開発技術。

上記のうち「説明を付して扱う知識」に該当する専門用語は、初出時に定義・解説を付す。国土交通省職員が IT 専門家の支援なく本ドキュメントの概要を把握できることを目標水準とする。

0.2　記載範囲と対象外

本節では、本ドキュメントが対象とする記載範囲と対象外の事項を明確にする。

本ドキュメントに記載すること：

- 各要素技術の選定背景・他の選択肢との比較・採用の根拠。
- 実証実験の仮説・検証フロー・結果と考察。
- 開発・運用を通じて得られた課題・改善アプローチ。
- 国土交通省職員（以下、国交省職員）からのフィードバックに基づく有用性評価。

本ドキュメントに記載しないこと：

- システム全体の機能仕様の網羅的な一覧（→ 要件定義書を参照）。
- API インターフェース・エンドポイント仕様（→ 要件定義書を参照）。
- 実装コードの詳細な解説（→ OSS 化されるソースコードを参照）。

0.3　本ドキュメントの構成

本ドキュメントは以下の章で構成される。

第 1 章　本実証の概要 本実証事業の背景・目的・課題認識・解決アプローチを述べる。行政データ活用における現状課題と、Veda が目指す価値を整理する。

第 2 章　システム構成の概要 Veda のアーキテクチャ構成・主要機能の概要・採用技術スタックの概説を述べる。Google Cloud・Amazon Web Services（AWS）・Microsoft Azure 等のクラウドサービスの利用構成を含む。各技術の詳細な設計背景と選定理由は第 3〜5 章に記載する。

第 3 章　データ構築モジュールにおける技術的工夫 システムの設計過程で、特筆すべき技術選定の背景と設計意図を記述する。メダリオンアーキテクチャの採用背景・DuckDB を活用したテーブルアクション高速化・UI/UX 改善の 3 点を中心に記載する。

第 4 章　データ構造化技術の調査と検証 非構造データ（PDF・紙書類）を AI で構造化するための OCR（光学文字認識）および LLM（大規模言語モデル）技術に関する調査・比較・検証結果を記載する。Azure Document Intelligence および AWS Bedrock 上の Claude 等の選定理由とプロンプトエンジニアリングの検証プロセス、非機能 KPI の定義と検証結果を詳述する。

第 5 章　データ活用モジュール（LINKS BI）における技術的工夫と性能検証 LINKS BI の AI チャット機能（Function calling・DuckDB-Wasm・Vega-Lite 等を組み合わせた自然言語データ分析）に関する技術検証の背景・手法・結果を記載する。アーキテクチャ設計の工夫と処理精度の性能検証を中心に記載する。

第 6 章　実証結果・有用性検証 実証を通じたユーザーフィードバックと機能評価を記録する。Veda について、有効に機能した点と改善が必要な点を観点別に整理する。技術的な詳細よりもユーザー視点の評価を中心に記載する。

第 7 章　ランニングコスト クラウドインフラ（Google Cloud・AWS・Azure）および外部 API（LLM・OCR 等）の実際の利用コスト実績と、運用を見据えたコスト最適化の知見を記載する。

第 8 章　成果と課題 本実証全体の成果のまとめと、残課題・今後の展望を記載する。

第 9 章　用語集 本ドキュメントで使用する専門用語・略語を一覧で定義する。

第 1 章　本実証の概要

本章では、Project LINKS の背景・目的・課題認識・解決アプローチを概説し、レポート全体の前提を整理する。

1.1　調査の背景・目的

本節では、Project LINKS の社会的背景と本実証の目的を整理する。技術的な詳細は第 2 章以降に委ねる。

1.1.1　Project LINKS の社会的背景

国土交通省をはじめとする行政機関では、EBPM（Evidence-Based Policy Making、エビデンスに基づく政策立案）の推進が求められており、政策判断の根拠となるデータを収集・整理・活用する能力が行政の重要な課題となっている。しかし、行政機関が日常業務で作成・蓄積する文書の多くは、紙帳票・PDF・スキャン画像など機械処理が困難な形式のままであり、データとして有効活用されていないのが実態である。

国交省の業務実態と課題

国土交通省では、様々な法的制度や予算事業等に基づき、民間事業者や自治体などから一定の様式で報告書や申請書類等を受け取り、その内容を集計して審査、実態把握、報告書作成、統計作成などに活用している。

しかし、これらの行政文書の多くは、紙帳票・PDF・スキャン画像・Microsoft Excel 等の非構造または半構造形式である。これらのデータは人間が閲覧・参照することを前提として作成されており、統計処理・横断集計・機械的な検索といった分析作業には直接利用できない。そのため現場では、紙や PDF で届いた書類を職員が手作業で Excel に転記し、転記後の Excel シートを集計してグラフを作成し、最終的に Word 等で定点観測レポートにまとめるという作業が広く行われている。こうした手作業には膨大な工数がかかるだけでなく、転記ミスや集計誤りといった品質上のリスクも伴う。一部の部門では独自システムを構築して集計作業を自動化しているケースもあるが、すべての部門がそのような環境整備を行うことは難しく、多くの現場では職員が人力で作業を続けている。

また、集計されたデータの分析・可視化にも課題がある。従来の BI ツールや統計解析ソフトウェアは、専門的な IT リテラシーとツール操作の習熟を前提としており、非エンジニアである国交省職員が日常業務の中で使いこなすことは難しい。大規模なデータを扱う場合にはサーバーインフラの整備が必要となることも多く、導入コスト・運用コストの面でも課題がある。

LINKS Veda は、こうした課題を技術的に解消することを目的として立ち上げられた。省内の部門が保有する非構造データを簡単に構造化して転記業務の省力化を実現し、さらにノーコードで資料作成や可視化ができる環境を提供。行政現場の業務負荷を抜本的に軽減するとともに、データを活用した政策立案の高度化（EBPM）を実現する基盤の構築を目指している。あ

わせて、構造化されたデータをオープンデータとして整備し、政策立案・研究・産業利用に資する形で提供することも目的としている。

1.1.2　LINKS Veda 開発の目的

2025 年度の LINKS Veda 開発プロジェクトでは、Veda を構成する 2 つのサブシステムの技術実証を通じ、非構造データの自動的な構造化処理と、これを活用した EBPM の実現に向けた進捗および課題を明らかにすることを目的としている。

Veda を構成する 2 つのサブシステムとは、「データ構築モジュール」および「データ活用モジュール（LINKS BI）」である。技術検証では、実際の行政データを利用して構造化処理の有効性を検証するとともに、データ分析・可視化基盤としての実用性を評価し、本格的な社会実装に向けた技術的知見を蓄積した。

1.2　課題解決のアプローチ

Veda の課題解決アプローチは 2 層の構造となっている。システム全体のデータフローを以下に示す。

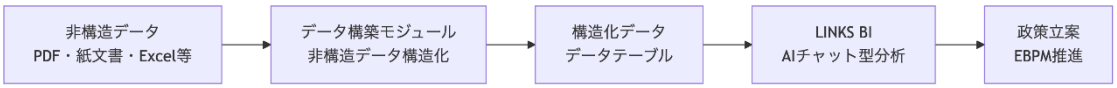


図 1 「データ構築モジュール」および「データ活用モジュール（LINKS BI）」によるデータ活用の全体フロー

1.2.1　「データ構築モジュール」によるデータ構造化基盤

「データ構築モジュール」は、LLM（大規模言語モデル）を活用した非構造データの自動データ構造化処理を中核機能とするサブシステムである。PDF や紙文書から意味情報を抽出し、あらかじめ定義したスキーマに従って構造化データを生成する。生成されたデータはデータテーブルとして管理され、G 空間情報センター（地理空間情報の流通を担うポータルサイト）へのオープンデータ公開や、LINKS BI での分析に利用される。データ品質を段階的に高めるアーキテクチャ設計についての詳細は第 3 章で述べる。

1.2.2　「データ活用モジュール（LINKS BI）」によるデータ分析基盤

「データ活用モジュール（LINKS BI）」は、自然言語による対話でデータ分析・可視化を実現する AI チャット型 BI アプリケーションである。ユーザーは専門的な SQL やプログラミングの知識を必要とせず、自然言語で問い合わせるだけでグラフやマップとして結果を得ることができる。非エンジニアである国交省職員でも、データ分析の成果を直接確認できる UI/UX を提供する。

技術的な特徴として、データ処理エンジン DuckDB の WebAssembly 版である DuckDB-Wasm（WebAssembly 版 DuckDB。WebAssembly とは Web ブラウザ上でネイティブに近い

速度でプログラムを実行する技術）を Web ブラウザ上で動作させる方式を採用しており、バックエンドサーバーと連携しつつも分析処理の大部分をブラウザ側で完結できる構成としている。これにより、サーバー側の分析負荷を抑えながらスケーラブルな処理が可能になる。詳細は第 5 章を参照すること。

1.2.3　システム全体の役割分担

「データ構築モジュール」および「データ活用モジュール（LINKS BI）」はデータの流れの中で上流・下流の関係にある。「データ構築モジュール」が非構造データを構造化データに変換し（上流）、「データ活用モジュール（LINKS BI）」がその構造化データを分析・可視化して政策判断を支援する（下流）という一貫したデータパイプラインを構成する。この 2 層構造により、「データを作る」課題と「データを使う」課題を同時に解決することが本プロジェクトのアプローチである。

1.3　創出価値

本実証を通じて創出される価値を、受益対象別に整理する。

政策的価値： 1.1 節で述べた紙文書・PDF の手作業による転記・集計という課題に対して、データ構築モジュールによる自動構造化により、データ収集・整理にかかる人的工数を大幅に削減できる。また、LINKS BI を活用したデータ分析の普及により、政策立案の品質向上やサイクルの短縮など、EBPM の実践を図る。

社会的価値： データに基づく政策立案の高度化により、行政サービスの質が向上し、限られた公共リソースの効果的な配分が可能になる。また、オープンデータとして整備されたデータセットは、民間や研究機関が二次活用できる社会インフラとして機能する。

本章で述べた課題認識と解決アプローチを前提として、第 2 章以降では具体的なシステム構成・技術選定・検証結果を詳述する。

第 2 章　システム構成の概要

本章では、Veda のシステム全体構成・主要機能の概要・採用技術スタックを整理する。各技術の詳細な設計背景と選定理由は第 3 章以降に記載する。

2.1　システムアーキテクチャと基盤構成

本節では、Veda の全体構成、各コンポーネントの役割、およびシステムが稼働するインフラストラクチャを整理する。

2.1.1　全体構成とデータフロー

Veda は「データ構築モジュール」と「データ活用モジュール（LINKS BI）」の 2 つのサブシステムで構成される。データ構築モジュールは非構造データ（PDF・Excel・画像等）を AI を活用して構造化するサブシステムであり、データ活用モジュール（LINKS BI）はデータ構築モジュールが出力した構造化データをブラウザ上でインタラクティブに分析・可視化するサブシステムである。

2 つのモジュールはデータの流れで連携する。ユーザーが非構造データをアップロードするところから始まり、AI による構造化処理、データ編集・加工、最終的な分析・可視化・公開という一連のワークフローを一体として提供する設計になっている。

2.1.2　データ構築モジュールのコンポーネント構成

Veda は 2 つのサーバーコンポーネントで構成される。

第 1 のコンポーネントは Web アプリケーションサーバーである。Remix（React フルスタックフレームワーク）を基盤として構築されており、静的ファイルの配信と主要な API 処理のすべてを担当する。アセット管理・データ編集・ワークフロー管理・メダリオンアーキテクチャ（第 3 章で解説）の各データレイヤー管理をこのコンポーネントが一手に担う。PostgreSQL でメタデータとユーザーデータを管理し、DuckDB を用いてデータ変換処理も行う。

第 2 のコンポーネントは非同期処理実行基盤（サーバーレス実行環境）である。Web アプリケーションサーバーから依頼を受け、AI による構造化処理・OCR（光学文字認識）処理・空間演算処理など、リクエスト/レスポンスのサイクルに収まらない重量級の処理を非同期で実行する。

2.1.3　データ活用モジュール（LINKS BI）コンポーネント構成

LINKS BI は API サーバーとブラウザ内処理で構成される。

API サーバーは AI チャット機能の中継処理とデータアクセス制御を担当する。AWS Bedrock の Claude モデルを呼び出して AI チャットの応答を生成し、HTTP ストリーミングでクライアントに結果を返す。認証・認可も API サーバーが担い、Veda の認証基盤と連携する形で実装されている。

ブラウザ内では、DuckDB-Wasm（WebAssembly 版 DuckDB）が直接動作し、データ分析を実行する。AI チャットの UI、ダッシュボードの可視化（Vega-Lite（JSON でグラフの内容を記述するだけで棒グラフ・折れ線グラフ等を生成できる可視化ライブラリ） ・MapLibre GL JS（ブラウザ上でインタラクティブな地図を表示するためのライブラリ））もブラウザ側で処理される。

2.1.4　システムインフラストラクチャ

本システムは Google Cloud を主要クラウド基盤とし、Amazon Web Services（AWS）および Microsoft Azure の特定サービスを組み合わせたマルチクラウド構成で運用される。インフラ全体のコンポーネントと接続関係を以下に示す。

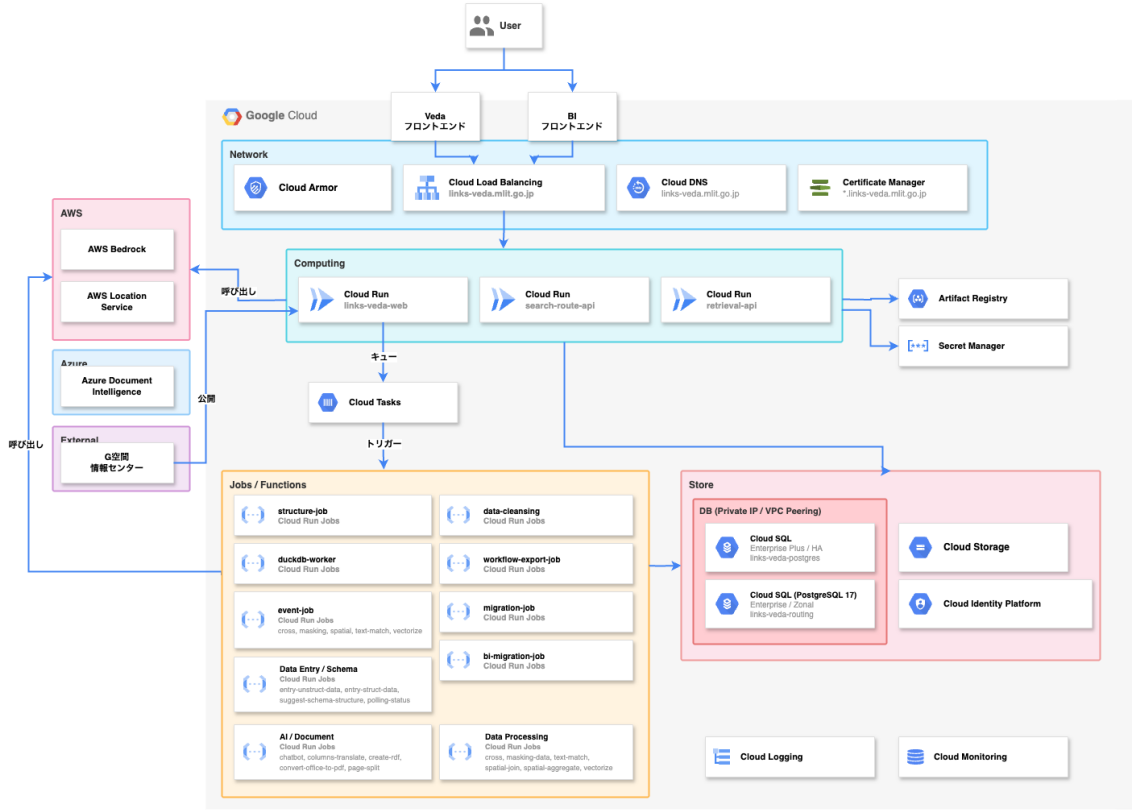


図 2 LINKS システム　インフラストラクチャ構成図

Google Cloud

フロントエンドの静的ファイル配信、バックエンドの API サーバー、非同期で行われるデータ構造化処理等、すべてのビジネスロジックは Google Cloud 上で稼働する。ファイルストレージには Google Cloud Storage（GCS）を利用し、ユーザーがアップロードしたアセットから加工済みの Parquet ファイルまで一元管理する。ユーザー認証基盤には Cloud Identity Platform を利用しており、メール/パスワード認証や OAuth 認証プロバイダーを提供する。Veda の両モジュールが同一の認証基盤を共有しており、認証連携を実現している。

Amazon Web Services（AWS）

AWS Bedrock は AI モデル実行環境として利用する。Anthropic 社製の Claude Sonnet モデルを呼び出すためのマネージド AI サービスであり、データ構造化処理・スキーマサジェッション・LINKS BI の AI チャット機能のすべてで Bedrock 経由で Claude モデルを使用する。

Microsoft Azure

Azure Document Intelligence は OCR 処理に使用する。PDF や画像アセットからテキストとレイアウト情報を高精度に抽出するために採用しており、データ構造化処理の前段処理として機能する。

2.2　データ構築モジュールの主要機能

Veda が提供する主要機能は、データ構造化機能、テーブルアクション・データ活用機能、およびオープンデータ化機能の 3 つに整理される。

2.2.1　データ構造化機能

データ構造化機能（データ構造化処理）は、非構造データ（PDF・紙書類・Excel 等）を機械可読な構造化データに変換する機能であり、以下の 3 つの機能で構成される。

- 第 1 の機能はスキーマサジェッションである。与えられた PDF 等のドキュメントを解析し、「どのような構造でデータを抽出すべきか」を AI が自動的に提案する。ここでスキーマとは、抽出する項目の名前・データ型・構造を定義したものである。ユーザーはこの提案をもとに抽出スキーマを確認・編集し、ワークフローとして保存する。
- 第 2 の機能はデータ構造化処理本体である。スキーマサジェッションで決定したスキーマに基づき、与えられた PDF 等のドキュメントから AI が指定の項目・型に従ってデータを抽出し、構造化されたデータテーブルとして出力する。
- 第 3 の機能はハルシネーション（AI が事実と異なる情報を生成する現象）防止のためのバリデーションである。LLM による抽出結果に対して複数回の評価を行い、抽出が失敗している可能性が高いフィールドや信頼度の低いフィールドに信頼度スコアを付与する。ユーザーはこのスコアをもとに、人手での確認・修正が必要な箇所を効率よく特定できる。

データ構造化処理本体は、OCR（光学文字認識）と AI（LLM・VLM）を組み合わせて非構造データから情報を抽出する。標準的な文書処理には OCR＋LLM 構成を適用し、打ち消し線や手書き修正など視覚的な解釈が必要な箇所には VLM（Vision-Language Model、視覚言語モデル）を併用する設計としている。抽出後のデータテーブルはユーザーが UI から直接編集でき、フィールドの型変更や値の修正も可能である。処理方式の選定経緯・技術比較・検証結果の詳細については第 4 章で述べる。

図 3 構造化結果確認画面

2.2.2　テーブルアクション機能

テーブルアクション機能は、構造化済みのデータテーブルに対してユーザーが UI から各種データ変換・加工操作を行うことができる機能である。

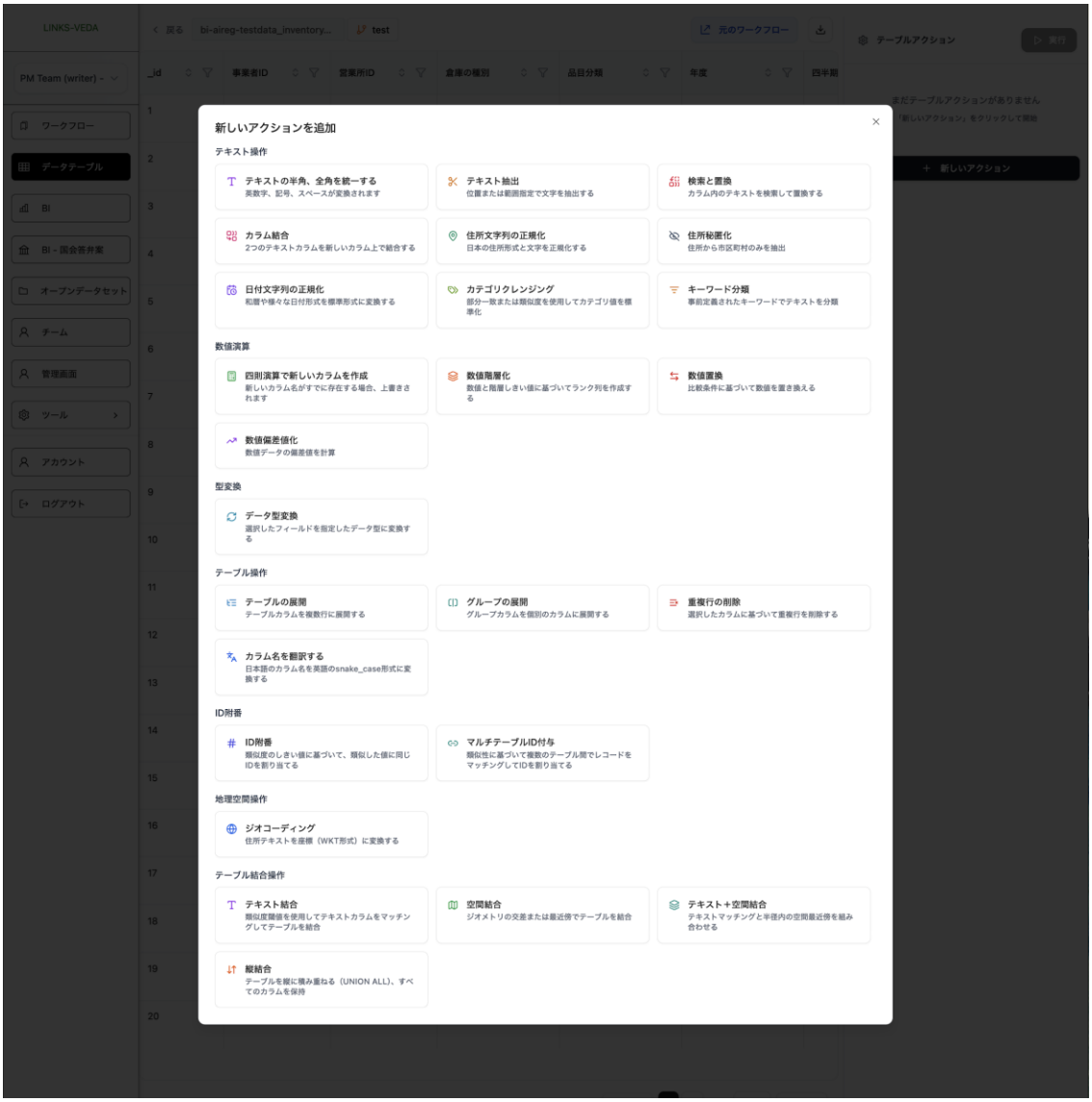


図 4 テーブルアクション機能（データ変換・加工の操作画面）

データの型変換・正規化・クレンジングから、日付の和暦対応変換・テキスト全角半角統一・住所正規化・類似度ベースの ID マッチング、ジオコーディング、テーブル結合（テキスト・空間・垂直方向）といった行政データに特化した処理まで、数多くのアクションを提供する。各アクションは DuckDB の SQL として実行され、処理結果は GCS 上の Parquet（高速な分析処理に適した列指向データ形式）ファイルとして更新される。アクションは一連のテーブルアクションとして保存されるため、新たな行が追加された際に同じ処理を再適用しやすい。

2.2.3 オープンデータ化機能

オープンデータ化機能は、構造化・加工済みのデータセットを CSV や Parquet 形式でエクスポートし、外部公開・配布できる状態にする機能である。Gold 層のデータセットから必要な形式でデータを出力でき、G 空間情報センター等への公開を想定している。あわせて、公開時に必要となるメタデータの生成・管理、および公開 URL の作成・管理も本機能に含まれる。

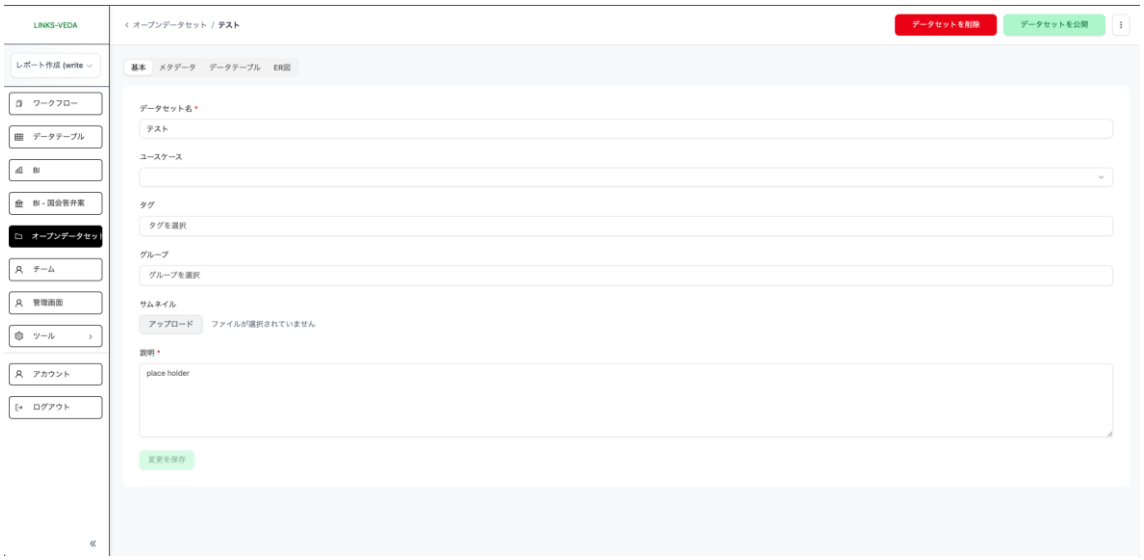


図 5 データセット画面

2.3 データ活用モジュール（LINKS BI）の主要機能

データ活用モジュール（LINKS BI）は、データ構築モジュールが構築したデータセットをブラウザ上でインタラクティブに分析・可視化するサブシステムである。AI チャット機能とダッシュボード機能の 2 つが中核を成す。

2.3.1 AI チャット機能

AI チャット機能は、データテーブルに対して自然言語で質問すると、AI が SQL クエリを生成・実行して結果をグラフや地図で表示する機能である。ユーザーは「都道府県ごとの件数を棒グラフで見せて」「緯度経度列を地図上にプロットして」のように日本語で指示するだけでデータの集計・可視化を行うことができる。

技術的には、API サーバーが AWS Bedrock の Claude モデルを呼び出し、BI アプリケーションに定義されたツール群（DuckDB での SQL クエリ実行・チャート生成・地図表示等）を Function calling（AI が必要に応じて外部ツールを呼び出す仕組み）で実行する構成になっている。レスポンスは Vercel AI SDK 独自の HTTP ストリーミング形式でクライアントに逐次配信される。詳細な技術検証結果については第 5 章で述べる。

図 6 AI チャット画面（入力画面）

図 7 AI チャット画面（結果表示画面）

2.3.2　ダッシュボード・可視化機能

ダッシュボード機能は、AI チャットで生成したグラフや地図を自由にレイアウトして 1 画面にまとめる機能である。複数のグラフ・地図・テーブルをグリッドレイアウト上に自由に配置してデータを探索・監視できる。Vega-Lite による宣言的チャート生成、MapLibre GL JS による地図表示を組み合わせて多様な可視化を実現する。データの分析処理はサーバーを介さずブラウザ内の DuckDB-Wasm で完結する設計のため、複数ユーザーが同時に分析しても互いに干渉しない高いスケーラビリティを持つ。

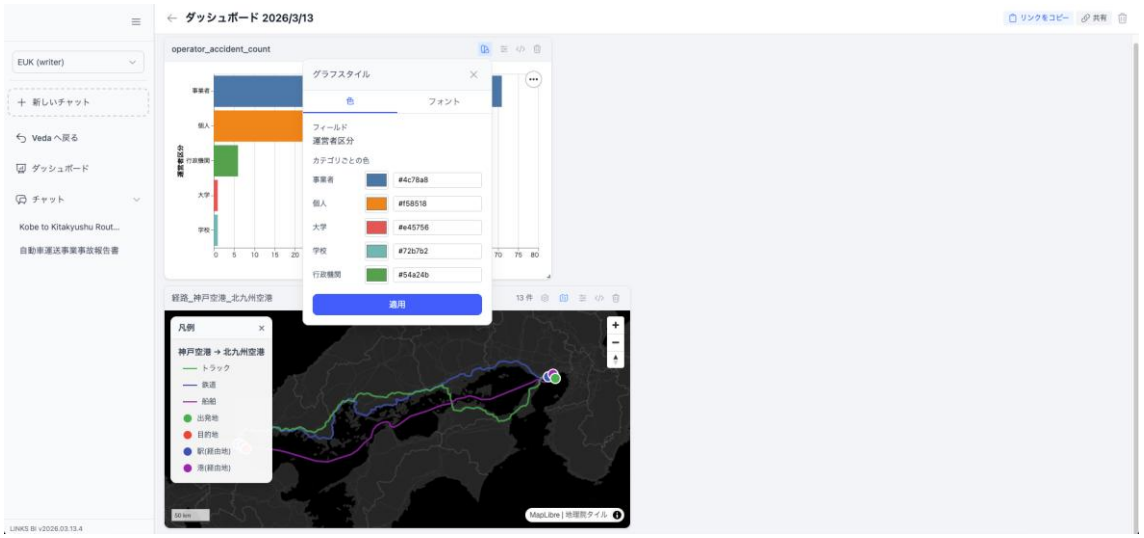


図 8 ダッシュボード・可視化機能

2.3.3 国会答弁案生成機能（個別機能）

国会答弁案生成機能は、ユーザーが入力したプロンプトに沿って答弁案を AI が自動生成する機能であり、汎用機能である LINKS BI に対し、その個別機能として構築している。Veda で構造化・整備されたデータに加えて、過去の国会答弁を参照しながら、目的に応じた答弁文の草案を生成する。過去答弁を活用した新規答弁案作成の作業負荷軽減を目的としている。

図 9 国会答弁案生成機能

2.3.4 その他の機能

AI チャットの延長として、データ分析やドキュメント検索に特化した複数の機能を提供する。これらはすべて AI チャットの Function calling 機構を通じて呼び出される形で実装されており、ユーザーが自然言語で指示するだけで適切な機能が AI によって選択・実行される。

提供する機能の一覧を以下に示す。

表 1　LINKS BI その他の機能一覧

機能	概要
経路検索	ネットワークデータを用いて最短経路や到達圏の算出を行う
回帰分析	変数間の相関・回帰モデルを自動生成し、結果をグラフで可視化する
クラスタ分析	K-Means アルゴリズム等を用いてデータを自動的にグループ化し、傾向の把握を支援する
ファイル検索	データ構築モジュールに登録されたドキュメント群に対してキーワードとセマンティック検索を組み合わせたハイブリッド検索で横断的に検索し、関連箇所を提示する
外部データ取得機能	e-Stat API からデータを取得して、人口データなどを自動的にチャットに読み込む
プリセット地図データ機能	地域メッシュデータや、都道府県ポリゴンを事前にアプリケーション内に内包することで、それらのポリゴンでの集計や可視化をする

図 10 その他のツール機能（例：経路探索）

2.4 採用技術スタック

本節では、 Veda の各モジュールで採用している主要な技術スタックの全体像と、主要ライブラリの選定根拠を整理する。各技術の詳細な設計背景は、データ構築モジュールついては第 3 章、データ構造化については第 4 章、LINKS BI については第 5 章でそれぞれ述べる。

2.4.1 データ構築モジュールの採用技術

バージョンはすべて執筆時点のものであり、随時更新される。

Web アプリケーションサーバー（Cloud Run）

Web アプリケーションサーバーで採用している主要技術を以下に示す。

表 2.2　Web アプリケーションサーバー採用技術

技術	バージョン	用途
Remix	v2.15	フルスタックフレームワーク（ルーティング・サーバーサイドレンダリング）
Hono	v4.6	Web サーバー／ルーティング基盤（Remix 連携）
React	v18.3	UI コンポーネント
TypeScript	v5.7	型安全な開発言語
Prisma	v6.3	PostgreSQL ORM
PostgreSQL（Cloud SQL）	16	リレーショナルデータベース（メタデータ管理）
DuckDB（node-api）	v1.4	サーバーサイドデータ変換・分析クエリ

非同期処理実行基盤（Cloud Run サーバーレス）— 主にデータ構造化処理

非同期処理実行基盤（Cloud Run サーバーレス）は Python で実装されており、データ構造化処理を中心とした非同期処理を担う。以下の技術を採用している。

表 2.3　非同期処理実行基盤採用技術

技術	用途
Azure Document Intelligence	OCR・テキスト・レイアウト情報抽出
Claude Sonnet 4.5（AWS Bedrock 経由）	LLM による情報抽出・信頼度評価・スキーマサジェッション
boto3	AWS Bedrock 上の Claude へのアクセス
langchain-aws / langchain-core	スキーマサジェッションの JSONOutputParser（LLM 出力のパース処理）
PyMuPDF	PDF 解析・ページ画像変換（VLM モード）
Pillow	画像前処理・リサイズ（VLM モード）

2.4.2　データ活用モジュールの採用技術

バージョンはすべて執筆時点のものであり、随時更新される。

フロントエンド（Web ブラウザ）

フロントエンドで採用している主要技術を以下に示す。

表 2.4　LINKS BI フロントエンド採用技術

技術	バージョン	用途
React	v19.2	UI コンポーネント
Vite	v7.2	ビルドツール
DuckDB-Wasm	v1.30	ブラウザ内分析エンジン
Vercel AI SDK	v6.0	AI チャット UI・ストリーミング処理
Claude Sonnet 4.5	—	LLM モデル（AWS Bedrock 経由）
TanStack Router	v1.141	クライアントサイドルーティング
TanStack Query	v5.62	データフェッチ・キャッシュ
jotai	v2.16	アトミックな状態管理
MapLibre GL JS	v5.15	地図表示
Vega-Lite	v6.4	宣言的チャート生成

API サーバー（Cloud Run）

API サーバーで採用している主要技術を以下に示す。

表 2.5　LINKS BI API サーバー採用技術

技術	バージョン	用途
Hono	v4.10	軽量 Web フレームワーク API サーバー
Kysely	v0.28	型安全な SQL クエリビルダー
PostgreSQL（Cloud SQL）	16	データベース（チャット履歴等）

2.4.3　主要ライブラリの選定根拠

LINKS BI の API サーバーおよびフロントエンドで採用している主要ライブラリについて、選定の背景と比較検討の結果を記述する。DuckDB および DuckDB-Wasm の選定背景は第 3 章

および第 5 章で、Vega-Lite の選定背景は第 5 章でそれぞれ詳述している。本項はそれ以外の主要ライブラリを対象とする。

Hono（LINKS BI バックエンドフレームワーク）

LINKS BI の API サーバーには Hono を採用した。Hono は Web Standards API（Fetch API）を基盤としたモダンな軽量 Web フレームワークである。バンドルサイズ（ Web アプリをブラウザに読み込む際に転送されるプログラムの合計サイズ。小さいほど初期表示が速い）が 14kB 未満・外部依存ゼロという軽量さと、TypeScript のエンドツーエンド型推論、そして HTTP ストリーミング配信のビルトインサポートが採用の主な理由である。AI チャットの中継処理という本 API サーバーの用途に適したシンプルな構成を実現している。

Vercel AI SDK / @ai-sdk/amazon-bedrock（AI ストリーミング・Function calling）

LINKS BI の AI チャット機能では、Vercel AI SDK（「ai」 v6.0・「@ai-sdk/amazon-bedrock」 v4.0）を採用した。

採用の背景：LINKS BI の AI チャットは、AWS Bedrock の Claude モデルを呼び出しつつ、複数ツールの Function calling をストリーミングで処理するという複雑な実装を要する。AWS SDK を直接使用する場合、Bedrock の Converse API のストリーミングレスポンス処理、Function calling のメッセージ履歴（ツール呼び出し→結果→次の LLM リクエスト）の管理、フロントエンドでのストリーミング受信と UI への反映、これらすべてを手動で実装する必要がある。

Vercel AI SDK はこれら一連の処理をパッケージとして提供しており、バックエンド側では ai-sdk 独自の HTTP ストリーミング形式によるレスポンス生成と Function calling の履歴管理を、フロントエンド側ではチャット形式の UI コンポーネントやストリーミング受信の仕組みを、それぞれ少ないコード量で実装できる。また、複数ツールを跨ぐエージェントループ（ツール呼び出し→結果フィードバック→次の判断）の管理を SDK が抽象化し、アプリケーション側の定型的なコード記述量を大幅に削減できた。バックエンドとフロントエンド間のプロトコル整合も SDK が吸収するため、通信仕様を独自に設計・実装する必要がない。これらの機能を一から開発する場合と比較して、開発工数を大幅に削減できた。また、LLM プロバイダー（AWS Bedrock・OpenAI・Google 等）を切り替え可能な統一 API を提供しており、将来のモデル変更にも柔軟に対応できる。

Kysely（LINKS BI バックエンド SQL クエリビルダー）

LINKS BI のバックエンドデータベースアクセスには Kysely（v0.28.0）を採用した。Kysely は TypeScript 向けの型安全な SQL クエリビルダーであり、ORM とは異なり SQL に近い API でクエリを記述できる。

採用の背景：LINKS BI のバックエンドはチャット履歴・ダッシュボード設定等の管理データを扱う PostgreSQL へのアクセスが中心である。Prisma のようなフル ORM はスキーマ駆動のコ

ード生成とマイグレーション管理が強力だが、生成される SQL が不透明で複雑なクエリの最適化が難しい側面がある。Kysely は「SQL 構文を TypeScript のメソッドチェーンで表現する」という設計思想のもと、API が SQL の構造に忠実であるため、実行される SQL が直感的に把握できる。公式ドキュメントによれば、Kysely の設計原則は「依存ゼロ」「すべてイミュータブル」「型安全の徹底」「SQL に近い API 設計」であり、コンパイル時にテーブル名・カラム名・JOIN 条件の型チェックが行われるため、スキーマ変更時の影響をコンパイルエラーとして検出できる。

MapLibre GL JS（地図表示ライブラリ）

LINKS BI の地図表示には MapLibre GL JS（v5.15.0）を採用した。MapLibre GL JS は WebGL を使用したベクタータイルレンダリングライブラリであり、GPU による滑らかな地図描画を実現するオープンソースライブラリである。

採用の背景： 地図表示ライブラリの候補としては CesiumJS も検討した。CesiumJS は 3D 地球儀表示・地形描画・建物モデル表示等の高度な 3D 機能を備えた強力なライブラリであるが、バンドルサイズが大きく（数 MB 規模）、ブラウザ上での初期読み込みや描画性能に影響を与えやすい。LINKS BI のユースケースでは、構造化データの地理的分布や統計値の地図上可視化が主な用途であり、3D 表示が必要となる場面はほとんどない。そのため、2D 地図表示に特化しシンプルかつ高速な描画が可能な MapLibre GL JS を選定した。MapLibre GL JS は BSD-3 条項ライセンスのオープンソースであり、API キーや利用料なしに商用利用できる点も利点である。スタイル表現は JSON 形式で宣言的に記述可能であり、AI チャットの Function calling で地図スタイルを動的に生成する本システムの用途とも高い親和性を持つ。以下の表は、MapLibre GL JS と CesiumJS との観点別の比較である。

表 2.6　MapLibre GL JS と CesiumJS の比較

評価観点	MapLibre GL JS	CesiumJS
主な用途	2D 地図（ベクタータイル）	3D 地球儀・地形
バンドルサイズ	軽量	大（数 MB 規模）
ライセンス	BSD-3（オープンソース）	Apache 2.0
描画性能（2D）	高速	本用途には不必要な機能が多い
スタイル定義	JSON（Mapbox Style 仕様）	独自 API

Jotai（フロントエンド状態管理）

LINKS BI のフロントエンド状態管理には Jotai（v2.16.0）を採用した。Jotai はアトミックな状態管理ライブラリであり、状態を最小単位に分割して独立に管理する設計思想を持つ。

採用の背景：LINKS BI のフロントエンドは、AI チャットの会話状態・データベースコンテキスト・チャート設定・地図スタイル・テーブル表示設定等、互いに独立した多数の状態を扱う。Redux のようなグローバルストア方式では定型的なコードの記述量が多く、状態の追加・変更に対して柔軟に対応しにくい。Jotai は各状態を独立した小さな単位として定義でき、状態間の依存関係も宣言的に表現できるため、機能の追加・変更に伴う状態の増減に対応しやすい。また、画面の各コンポーネントが必要な状態のみを参照する仕組みにより、無関係な状態変化による画面の不要な再描画を抑制でき、パフォーマンス面でも有利である。

本章では、データ構築モジュールとデータ活用モジュール（LINKS BI）の 2 つのサブシステムから成る Veda の全体構成を整理した。データ構築モジュールが非構造データの変換を担い、データ活用モジュール（LINKS BI）がその構造化データの分析・可視化を担うという基本構造が、以降の章で述べる技術的検証の前提となる。各技術の詳細な設計背景と選定理由は、データ構築モジュールについては第 3 章、データ構造化については第 4 章、LINKS BI については第 5 章にそれぞれ記載している。

第 3 章　データ構築モジュールにおける技術的工夫

本章では、データ構築モジュールの設計において特筆すべき技術的選定の背景と設計意図を記述する。データパイプライン設計、サーバーサイドでの DuckDB の活用、UI/UX 改善の 3 点を取り上げる。

3.1　データパイプライン設計（メダリオンアーキテクチャ）

本節では、Veda のデータパイプライン設計の根幹であるメダリオンアーキテクチャと、その採用を支える技術的概念について解説する。

3.1.1　データエンジニアリングとその課題

データエンジニアリングとは、組織が保有する大量のデータを分析・活用できる状態に整備するための技術・設計の体系である。近年の機械学習・AI の普及により、センサーデータ・業務データ・画像・テキストなど多種多様なデータが大量に生成されるようになった。これらの生データをそのまま分析に使うことはできず、不整合の除去・型の統一・欠損値の処理・結合・集計等の変換処理が必要となる。データエンジニアリングはこうした「データを分析可能な状態に変換するパイプライン」を設計・構築・運用する技術領域である。

行政データ管理でも同様の課題がある。紙文書・PDF として蓄積されてきた非構造データをデジタル化し、複数部署・複数年度のデータを統合して分析・政策判断に活用するためには、体系的なデータパイプラインの設計が不可欠である。

行政データ特有の課題として、品質管理の難しさがある。非構造データから抽出した値には誤認識・表記揺れ・欠損が含まれることが多く、これらを一括処理するのではなく段階的に整理・検証しながら加工していく必要がある。また、ユーザーによる手動修正・追記も想定されるため、途中段階のデータを保持しつつ最終的な分析用データセットを構築できる仕組みが求められる。こうした課題に対応するために、本システムではメダリオンアーキテクチャを採用した。

3.1.2　行指向と列指向：データ格納モデルの選択

データをどのように格納するかは分析処理の速度に大きく影響する。行政データの分析では「列指向（カラム指向）」のデータ格納モデルが特に有効であるため、以下にその概念を解説する。

行指向（Row-Oriented）とは、データを 1 行ずつ連続して格納するモデルである。レコードの挿入・更新・個別レコードの検索に優れており、業務システムのトランザクションデータベース（PostgreSQL・MySQL 等）で一般的に採用される。たとえば、ユーザーが 1 件のデータを登録・編集する操作は行指向データベースが得意とする処理である。

列指向（Column-Oriented）とは、同じカラムのデータをまとめて連続して格納するモデルである。特定のカラムだけを集計・フィルタリングする分析クエリに対して、行指向より大幅に高速に処理できる。

たとえば、次の 3 行 3 列データを考える。

表 2.7　行指向と列指向の格納例

都道府県	年度	損傷件数
東京都	2023	120
大阪府	2023	95
福岡県	2023	80

行指向では、「[東京都,2023,120] [大阪府,2023,95] [福岡県,2023,80]」 のように 1 行単位で連続して格納される。したがって「損傷件数」だけを集計したい場合でも、各行の他カラム（都道府県・年度）を含めて読み込む必要がある。

列指向では、「都道府県=[東京都,大阪府,福岡県]」、「年度=[2023,2023,2023]」、「損傷件数=[120,95,80]」 のように列単位で格納される。このため「損傷件数」の合計を計算する際は「損傷件数」 列だけを読み込めばよく、読み取り量を削減できる。また、同じ型・近い値のデータが連続して並ぶため圧縮効率も高くなる。

列指向データを格納するファイルフォーマットの代表例として、Apache Parquet が挙げられる。Parquet は、Apache Software Foundation が管理するオープンソースの列指向ファイルフォーマットであり、Hadoop・Spark・DuckDB をはじめとする主要なデータ処理エンジンが標準的にサポートしている。データを列単位で圧縮・格納するため、分析クエリに対して高い I/O 効率と圧縮率を実現できる。近年のデータエンジニアリング分野では、分析用データの永続化フォーマットとして Parquet が事実上の第一候補となっており、本システムの Gold 層でも Parquet を採用している（詳細は後述「本システムにおける実装」を参照）。

行政データ分析では「特定の年度の道路損傷件数を都道府県別に集計する」「複数年度の修繕費用の推移を分析する」といった列方向の集計処理が中心となるため、列指向モデルが特に有効である。

3.1.3　メダリオンアーキテクチャの概要

メダリオンアーキテクチャ（Medallion Architecture）とは、Databricks 社が提唱したデータ品質管理のためのデータレイヤー設計パターンである（参考: Databricks Glossary - Medallion Architecture）。データを品質レベルに応じて Bronze・Silver・Gold の 3 層に分けて管理し、各層で段階的にデータを精製していく考え方であり、「マルチホップアーキテクチャ」とも呼ばれる。

Bronze 層（生データ層）はデータソースから取り込んだ生データをそのまま保管するレイヤーである。変換・加工は行わず、元のデータを忠実に保持する。データの再処理が必要になった場合に原本として参照できる。

Silver 層（クレンジング済みデータ層）は Bronze 層のデータをクレンジング・型変換・正規化した状態で保管するレイヤーである。重複の除去・欠損値の処理・スキーマの統一などが行われ、扱いやすい形に整備されている。

Gold 層（分析用データ層）は Silver 層のデータを集計・加工して分析や機械学習に最適化した状態で保管するレイヤーである。ダッシュボード・AI チャット・外部公開など具体的なユースケースに合わせて整形された最終成果物である。

この 3 層構造により、各層のデータを独立して管理・検証できる。また特定の層から再処理が必要になった場合でも、上位層のデータを再生成することが容易である。

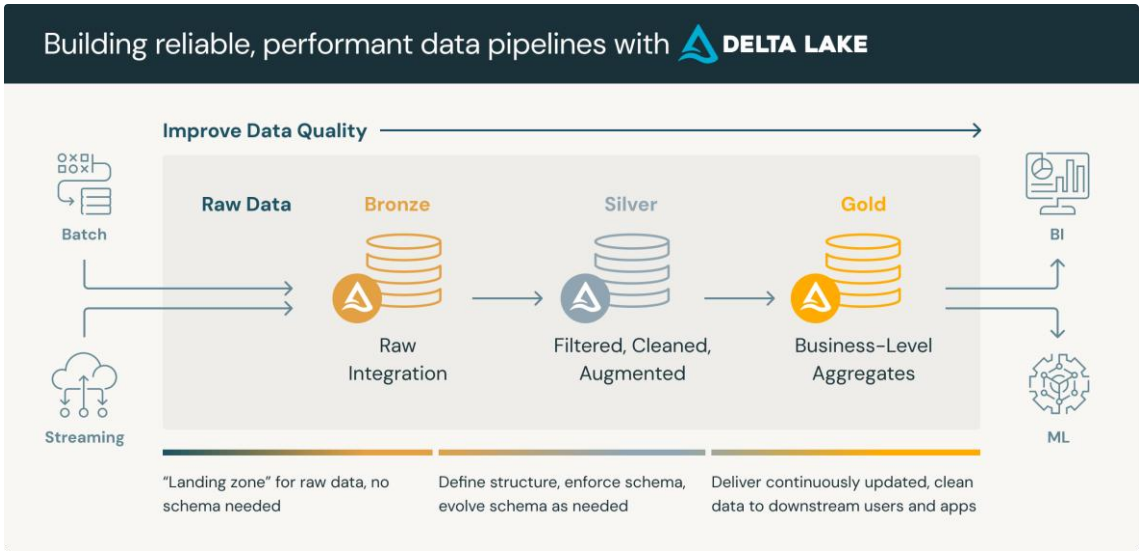


図 11 メダリオンアーキテクチャ概要（出典: Databricks, Medallion Architecture）

3.1.4 本システムにおける実装

現行のデータ構築モジュールの Silver 層は、PostgreSQL と GCS の 2 つのストレージで構成される。データ構造化処理の結果（構造化データ本体）は PostgreSQL に格納され、ユーザーによる構造化結果の編集も PostgreSQL 上で行われる。加えて、ワークフロー実行ごとの履歴を Parquet スナップショットとして GCS に保存し、スナップショットのメタデータも PostgreSQL で管理する。Silver 層の構造化データをデータテーブルとしてエクスポートすると Gold 層（Parquet）に出力される。テーブルアクション（データ変換・加工処理）はこの Gold 層のデータに対して実行され、処理結果は新しい Gold 層のデータとして生成される。以下に本システムにおける各層の実装を示す。

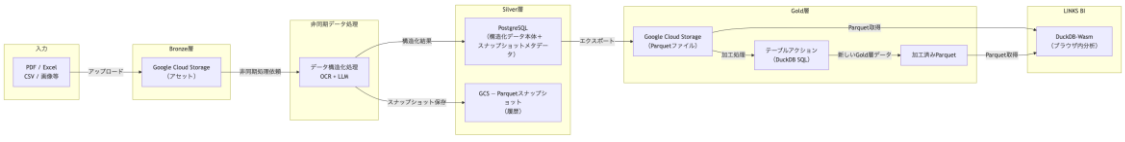


図 12 データ構築モジュールのメダリオンアーキテクチャ実装

Bronze 層：GCS（アセット）

ユーザーがアップロードしたアセットはそのまま GCS に保存される。PDF は PDF のまま、Excel は Excel のまま、加工せずに保持する。データ構造化処理の入力はこの Bronze 層のアセットを参照する。データテーブルへのインポートとしてアップロード Bronze 層のデータはサーバーサイドの DuckDB にもインポートされ、後続の AI 処理・クレンジング処理のためのデータソースとして機能する。

Silver 層：PostgreSQL + GCS Parquet（構造化データ）

Silver 層は PostgreSQL と GCS の 2 つのストレージで構成される。データ構造化処理の結果（構造化データ本体）は PostgreSQL に直接格納され、ユーザーが UI から行う構造化結果の編集も PostgreSQL 上のデータに対して実行される。あわせて、ワークフロー実行ごとの履歴を Parquet スナップショットとして GCS に保存し、スナップショットのメタデータ（バージョン番号・スキーマ定義等）も PostgreSQL で管理する。

Gold 層：Parquet ファイル（分析最適化データ）

Silver 層の構造化データをデータテーブルとしてエクスポートすると、Parquet 形式へ変換され Gold 層として GCS に出力される。この Gold 層の Parquet ファイルが LINKS BI のデータソースとなる。

テーブルアクション（データ変換・加工処理）はこの Gold 層の Parquet データに対して DuckDB SQL で実行される。処理結果は Silver 層に戻すのではなく、新しい Gold 層の Parquet ファイルとして生成される。すなわち、加工のたびに Gold 層内で新しいデータが派生する設計であり、元の Gold 層データを上書きしない。

Gold 層に Parquet を採用した理由は以下の 3 点である。

- 第 1 に、Parquet は列指向フォーマットであり、同じ型・近い値のデータが連続する特性によって高い圧縮効率を得られるため、行政データのようなテキスト・数値混在データでもサイズ削減効果が大きい。
- 第 2 に、列指向の格納方式により特定列の集計・フィルタリング処理を効率化できるため、DuckDB-Wasm がブラウザ上で Parquet ファイルを直接読み込んで実行する分析クエリと相性がよい。
- 第 3 に、Parquet をイミュータブルな静的ファイルとして GCS に配置すると、同時アクセス時にデータベース接続数やロック競合の影響を受けにくく、LINKS BI のブラウザ内 DuckDB-Wasm が個別取得して分析できるスケーラブルな構成を実現できる。

Silver 層実装の変遷（課題と対策）

Silver 層の管理基盤の選定にあたっては、データレイクをテーブルとして運用するフレームワークを検討した。DuckLake は、データ本体を Parquet 等のオブジェクトストレージ上に保持しつつ、テーブルのスナップショット・スキーマ・変更履歴などのメタデータを管理してテー

ブルとして扱うための仕組みである。こうした「データレイクをテーブルとして運用する」枠組みは一般にオープンテーブル形式／レイクハウス系フレームワークとして整理され、代表例として Apache Iceberg、Delta Lake、Apache Hudi がある。

Silver 層は当初、この DuckLake を参考にしたアーキテクチャとして実装していた。

DuckLake が内部で差分を Parquet として生成・管理する方式を採用しているため、本システムでも変更・削除を含む差分を都度 Parquet ファイルとして蓄積し、Gold 層への出力時に差分ファイル群を Compaction（複数の差分ファイルを統合してファイル数を削減する処理）して 1 つの出力にまとめる方式としていた。

しかし、変更量が増加して差分ファイルが数千〜数万件規模になると、Gold 層への書き出し時にそれらのファイルを一度ストレージから読み出して統合する必要があり、I/O オーバーヘッド（ストレージへのデータ読み書きに伴う余分な処理時間）が急増した。この方式では処理時間と運用負荷が大きくなり、実用運用が困難になった。

この課題への対策として、差分ファイルの継続的な蓄積方式を見直した。現在は構造化データ本体を PostgreSQL に直接格納し、ワークフロー実行ごとの履歴を 1 スナップショットファイルとして GCS に書き出す方式を採用している。構造化結果の編集は PostgreSQL 上で行い、スナップショットのメタデータ管理も PostgreSQL で行うことで、差分ファイルの再統合が不要となり、I/O 負荷を回避し更新処理の安定性を改善している。

3.2　サーバーサイド DuckDB によるテーブルアクション

データ構築モジュールでは、サーバーサイドでもデータ変換・集計処理に DuckDB を活用している。データ構築モジュールが提供するテーブルアクション機能（27 種類のノーコードデータ処理）は、内部的に DuckDB SQL として実行される。

DuckDB はインメモリ（データをハードディスクではなくメモリ上に読み込んで高速処理する方式）の列指向 OLAP エンジンであり、集計・フィルタリング・型変換等の分析的な処理を PostgreSQL 等の行指向 OLTP データベースと比較して高速に実行できる。テーブルアクションでは、Gold 層の Parquet データに対してユーザーが UI 上からノーコードで指定した処理内容を DuckDB SQL に変換し、サーバーサイドの DuckDB インスタンス上で実行する。処理結果は新しい Gold 層の Parquet ファイルとして GCS に出力され、LINKS BI のデータソースとして利用できる。

テーブルアクションとして提供される処理の例を以下に示す。

- 型変換：カラムのデータ型を文字列・数値・日付等に変換する。
- 正規化：値の表記を統一的な形式に揃える。
- クレンジング：不正値・異常値の検出と除去を行う。
- 和暦対応変換：和暦表記の日付を西暦に変換する。
- テキスト全角半角統一：全角文字と半角文字の混在を統一する。
- 住所正規化：住所表記の揺れを標準形式に正規化する。
- ID マッチング：異なるデータセット間で ID を照合・紐付けする。

- ジオコーディング：住所文字列から緯度・経度を取得する。
- テーブル結合：複数のテーブルを指定したキーで結合する。

これらの処理を DuckDB SQL として実行すると、行指向データベースでは処理時間が長くなりがちな大量データに対するカラム単位の変換・集計処理を効率的に実行できる。ユーザーは SQL の知識を必要とせず、UI 上の操作のみでデータの加工・整備を完結できる。

3.3　データ構築モジュールにおける UI/UX の課題と改善

2024 年度の実証実験を通じて明らかになった課題を踏まえ、2025 年度の開発ではデータ構築モジュールのアーキテクチャとソースコードをゼロから再設計・再実装した。2024 年度の設計思想や実証で得られた知見は引き継ぎつつも、システム基盤そのものを刷新することで、UI/UX・性能・拡張性の各面で抜本的な改善を図った。以下では、2024 年度に確認された主要な課題と、それに対する 2025 年度の改善内容を整理する。

3.3.1　2024 年度の課題

2024 年度の LINKS Veda では、システムの操作性に関する複数の課題が確認された。以下の 4 点が主要な課題であった。

ワークフロー概念の難解さ

2024 年度の UI では、データ構造化・テキストマッチング・クロス集計・ジオコーディング等の各処理ステップが「オペレーター」として画面上に並列に表示されていた。初めてシステムを触るユーザーにとっては、それぞれのオペレーターが何を行うものかが画面上の表示だけでは把握できず、次にどの操作を行えばよいかがわからないという問題があった。

図 13 2024 年度のオペレーター画面

多くの場合、ユーザーはまずデータ構造化処理を行い、その後にテキストマッチングやクロス集計等のデータ加工処理を順に実行して最終的な分析用データを作成する。しかし 2024 年度の UI ではこれらのステップ間に前後関係や順序の案内がなく、ユーザーはどのような手順に沿って処理を進めれば正しいデータを作成できるかが画面からは判断できなかった。結果として、途中で操作に迷い利用を断念するケースが多く発生した。

スキーマ設定の複雑さ

2024 年度の UI では、構造化処理に先立って抽出対象のデータスキーマ（各フィールドの名前とデータ型）をユーザーが一つ一つ手動で定義する必要があった。

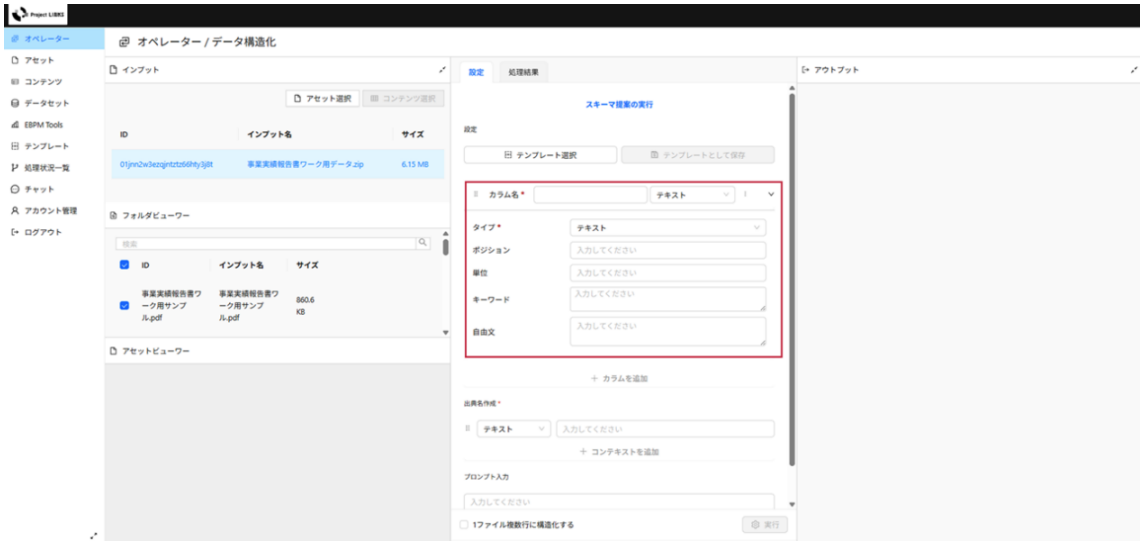


図 14 2024 年度のスキーマ設定画面

しかし、ノンエンジニアの国交省職員にとっては、「文字列型」「数値型」「日付型」といったデータ型の概念を理解しながらデータ構造を決定すること自体が難解であった。加えて、行政帳票では抽出すべきフィールド数が数十項目に及ぶことも多く、それらを一つ一つ手動定義する作業は現実的にユーザーが操作できる範囲を超えていた。

構造化結果の検証手段の不足

構造化処理の完了後、抽出結果の妥当性を確認するためにはユーザーが別途元の PDF ファイルを開き、抽出された値と原本を目視で照合する必要があった。システム内に抽出結果と原本を対比する手段が用意されていなかったため、確認作業に大きな工数がかかっていた。

抽出形式の制限

キーバリュー形式（1 つのキーに対して 1 つの値）の抽出のみに対応しており、1 枚の書類に複数の表が埋め込まれるケースや、1 つのキーに対して複数行・複数列のテーブル形式で値が対応するケースへの対応が困難であった。

3.3.2　2025 年度の実施改善

2025 年度ではアーキテクチャをゼロから再設計し、上記の課題に対して以下の改善を実施した。

スキーマサジェッションの自動化

PDF アップロード時点で AI が文書の内容を解析し、抽出スキーマの候補を自動的に提案する機能（スキーマサジェッション）を導入した。ユーザーはゼロからスキーマを定義する必要がなくなり、AI が提案したスキーマを確認・編集するだけでよい。あわせて、「この文書を何の

ために使うか」「どの情報を重点的に取り出したいか」を答えるアンケート形式フォームを追加し、回答内容をもとに AI へのプロンプトをカスタマイズして提案精度を向上させた。この改善により、ワークフロー概念の難解さとスキーマ設定の複雑さの双方が解消された。

図 15 ワークフローのアンケート形式フォーム

バウンディングボックス表示による構造化結果の検証

構造化結果の確認画面において、元の PDF（処理対象の非構造データ）と抽出結果（構造化されたフィールド一覧）を左右に並べて対照表示する機能を実装した。さらに、各フィールドの値が元 PDF 上のどの領域から抽出されたかをバウンディングボックス（矩形ハイライト）で視覚的に示す機能を追加した。ユーザーは別途 PDF を開くことなく、画面内で原本と抽出結果を対比しながら目視でのデータ確認・レビューを行うことができる。

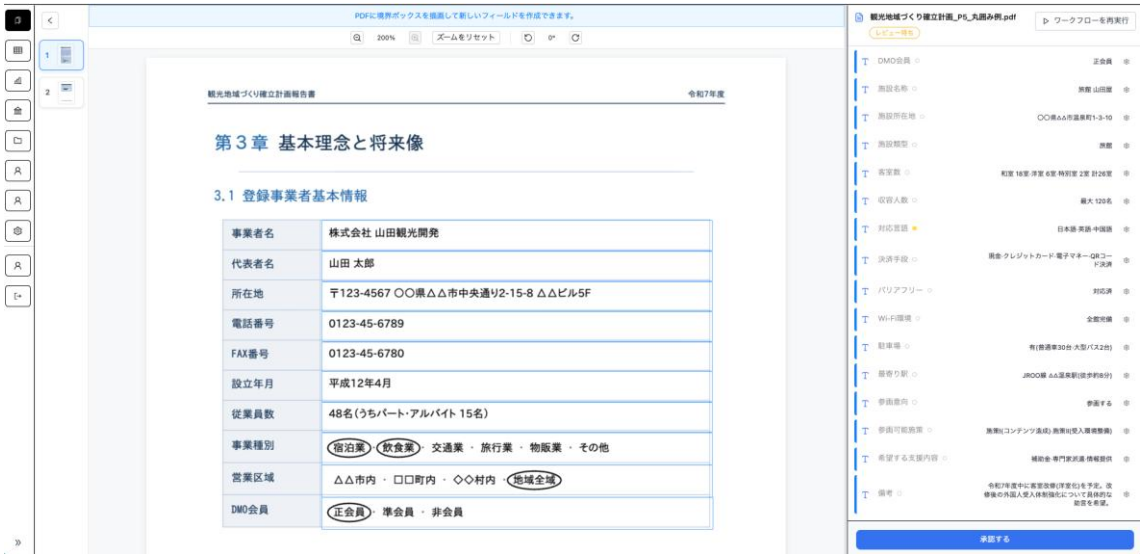


図 16 構造化結果の確認画面

テーブル形式の抽出対応

1 つのキーに対応する値が複数行・複数列のテーブル形式になり得る書類でも、表の構造を保ったまま抽出できるようにした。これにより、行政帳票に頻出する複合表形式の書類にも対応可能となった。

3.4　本章のまとめ

本章ではデータ構築モジュールの 3 つの技術的工夫を述べた。メダリオンアーキテクチャによる段階的なデータ品質管理、DuckDB によるテーブルアクションの高速化、UI/UX の再設計により、行政データの構造化から加工・分析までを一貫して支える基盤が整備された。データ構造化処理そのものの技術的な調査と検証については、次章（第 4 章）で詳述する。

第 4 章　データ構造化技術の調査と検証

本章では、行政文書の「データ構造化処理」を対象とし、社会実装に耐えうる水準へ発展させるため、実施した技術調査および検証の経緯・方法・結果を整理する。まず 4.1 節で調査の背景と目的を述べる。4.2 節で技術検証の出発点となった 4 つの課題（A〜D）を整理する。続いて 4.3〜4.6 節で各課題について「仮説→手法→計測方法→検討した選択肢→結果→結論」の順にまとめる。4.7 節では各検証の結論を統合し、推奨する技術構成を示す。4.8〜4.9 節で最終構成を評価するための KPI と検証結果を整理し、4.10 節で残る課題と今後の取り組みをまとめる。

4.1　背景と目的

本調査は、実運用や大規模展開でも安定して稼働するデータ構造化技術を選定し、その性能を最大限に引き出す処理アーキテクチャを整備することを目的として実施した。具体的には、OCR モデル、VLM、LLM、関連ライブラリ、アーキテクチャの候補を比較評価し、本システムに最適な技術構成を決定した。評価軸は「精度」「速度」「コスト」「運用性」「セキュリティ」の 5 つに統一し、同一条件で比較を行った。このうち、「セキュリティ」は国内リージョンで利用可能なこと、かつ、 ISMAP（政府情報システムのためのセキュリティ評価制度）対応可能なこととし、本条件に当てはまる構成を選定対象とした。

調査設計では、2024 年度の実証で得られた知見を出発点とした。2024 年度には、データ構築モジュールを用いた PDF 行政文書の OCR＋LLM 構造化の初期実証を実施し、様々なテーマで行政文書のデータ化及び分析を行うユースケースにより有効性を確認した。一方、精度・処理速度・大規模文書対応に課題が残った。本年度は、打ち消し線検知の二段判定、Excel 図形の構造化、分割戦略、プロンプト最適化、並列処理など、これまでの検討で有効性や課題が確認された要素を整理し、再現性のある比較と意思決定ができる形に再構成した。あわせて、本調査の結果として、今年度の実装項目と継続検証・将来検討項目を切り分け、今後の取り組みの見通しをまとめた。

4.2　課題の全体像

本節では、検証の出発点となった課題を整理する。課題は大きく「構造化精度」「大規模文書処理」「スケーラビリティ」「オフィスファイル対応」の 4 領域に分類され、発見の経緯により「2024 年度業務から明らかになったもの（課題 A〜C）」と「今年度の実証を進める中で新たに明らかになったもの（課題 D）」に分かれる。

4.2.1　2024 年度業務から明らかになった課題

昨年度は、本システムによるデータ構造化を複数のユースケースで実証した。その結果、主に以下の技術的課題が残った。

課題 A：構造化精度

本課題は、以下の 2 つの小課題を含む。いずれも OCR による文字抽出だけでは解決できず、視覚的コンテキストを含めた処理が必要であるという共通点を持つ。

- 図形要素の解釈：　丸囲みや矢印などの図形要素を、AI が意図どおりに解釈できないケースが多く発生した。人間にとって意味のある図形が、AI には単なる線として扱われ、選択肢の読み取りや要素間の関係理解に失敗することがあった。
- 打ち消し線・手書き修正の処理：　打ち消し線や手書きによる上書き修正などのイレギュラー記載に十分に対応できていなかった。OCR が訂正前後の文字列を区別できないため、誤情報や無効な値が混在する事例が確認された。

これらの結果から、文字抽出だけではなく、図形や修正痕といった視覚的コンテキストを含めた処理アプローチが必要であると結論づけた。

輸送実績（前年4月1日から3月31日まで）

	延実在車両数（日車）	延実働車両数（日車）	走行キロ（キロメートル）	実車キロ（キロメートル）	輸送トン数		営業収入（千円）
					実運送(トン)	利用運送(トン)	
北海道運輸							
東北運輸局							
北陸信越運輸局	9000 ~~11705~~						
関東運輸局							
中部運輸局							
近畿運輸局							
中国運輸局							
四国運輸局							
九州運輸局							
沖縄総合事務局 運							
全国計	9000 ~~11705~~						

図 17 打ち消し線を含む帳票例

輸送実績（前年4月1日から3月31日まで）

	延実在車両数（日車）	延実働車両数（日車）	走行キロ（キロメートル）	実車キロ（キロメートル）	輸送トン数 実運送（トン）	輸送トン数 利用運送（トン）	営業収入（千円）
北海道							
東　北							
北陸信越							
関　東							
中　部							
近　畿							
中　国							
四　国							
九　州							
沖　縄							
全国計							

図 18 打ち消し線のない帳票例

2－1. 格付情報（EEDIを用いる場合）

計算値	
基準値	
改善率【%】	
格付評価	

2－2. 格付情報（代替手法を用いる場合）

申請手法	建造設計段階 ・ 水槽試験結果 ・ 海上公試運転結果 ・ 実運航時の試験結果
代替手法の計算値	
代替手法の基準値	
改善率【%】	
格付評価	★★★★★

2－3. 格付情報（「第4　基準値を持たない船種の格付」として申請する場合（暫定運用手法））

対策内容	
申請船のCO2排出量	
比較船のCO2排出量	

図 19 丸囲みを含む Excel 帳票

2－1. 格付情報（EEDIを用いる場合）

計算値	
基準値	
改善率【%】	
格付評価	

2－2. 格付情報（代替手法を用いる場合）

申請手法	建造設計段階 ・ 水槽試験結果 ・ 海上公試運転結果 ・ 実運航時の試験結果
代替手法の計算値	
代替手法の基準値	
改善率【%】	
格付評価	☆☆☆☆☆

2－3. 格付情報（「基準式を持たない船種及び基準式の適用範囲外の船舶の格付（暫定運用手法）」として申請する場合）

申請方法	(1) ・ (2) ・ (3)

図 20 丸囲みを含む Excel 帳票

課題 B：大規模ドキュメントの処理

数百ページに及ぶ行政文書では、LLM の入力トークン（LLM が一度に考慮できる文字数）上限を超えて処理が失敗するケースが頻発した。機械的にチャンク分割しても、文脈が分断されて正確な情報抽出が難しくなることがあり、大規模文書でも破綻しない構造化アーキテクチャと処理設計が求められた。

課題 C：スケーラビリティ

処理時間が実務上のボトルネックとなった。2024 年度時点では、データ構造化処理において同時並行で処理ができるファイル数はシステム全体で数十件程度が上限であり、かつデータ構造化処理 1 回につき 1 ページあたり平均 45 秒の処理時間を要していた。これは、たとえば 100 ページの文書 1 件を処理するのに約 75 分かかる計算であり、1 日に数十件の文書を処理する実務には到底耐えられない水準であった。これはとくに LLM の API レートリミット（決められた時間内に利用できる API のリクエスト制限）や再処理の発生が効率を低下させていた。実運用では、大量ファイルを並列に処理し、限られた時間内でスループット（単位時間で処理できるファイル数）を大幅に向上させる必要があるため、アーキテクチャと制約条件の整理が必要となった。

4.2.2　実証を進める中で明らかになった課題

課題 D：オフィスファイルの処理

さらに、実証を進める中で、構造化対象には PDF だけでなく、Word、Excel、PowerPoint などの Office ファイルが含まれることが明らかになった。これらを LibreOffice（オフィルファイルから PDF への変換ツール）等で PDF 変換してから処理しようとすると、フォントや図形レイアウトの互換性の問題により、変換後 PDF で図形がずれたり、テキストが欠落したりする品質劣化が発生した。さらに、Excel を PDF 化するとテーブルがページをまたいで分割され、LLM が前後関係を把握できず、正確な構造化が難しくなるケースも確認された。このため、オフィルファイルの構造化については、単純に PDF 変換して構造化するとはのとは異なる処理手順を確立する必要があることがわかった。

4.3　課題 A：構造化精度の向上

本節では、課題 A（構造化精度）に対する検証を「仮説→手法→計測方法→検討した選択肢→結果→結論」の順で整理する。課題 A は複合的であり、単一の検証では解決できない。そのため、本節では 4.3.1 に示す 5 つの仮説を、4.3.2 で共通の評価方法と計測指標を定義する。4.3.7 ではサブ検証の結果を踏まえたモデル自動選定の検証を、4.3.8 で課題 A 全体の結論をまとめる。

4.3.1　仮説

課題 A 全体に対し、以下の仮説とサブ検証項目を設定した。各サブ検証は共通の評価指標（4.3.2）を用いつつ、サブ検証ごとに完結する構成としている。

表 4.1　構造化精度に関する仮説とサブ検証項目一覧

仮説	仮説内容	対象	節
① OCR 構成	VLM 単体より、OCR と LLM を組み合わせた構成の方が安定した精度を発揮する。OCR は文字パターンの認識に特化しており、かすれ	OCR＋LLM vs	4.3.3

	や筆跡のばらつきを含む手書き文字でも安定したテキスト変換が可能であるためである。	VLM 単体の基本構成比較	
②打ち消し線	打ち消し線を含むフィールドでは、画像全体を処理できる VLM の方が OCR+LLM 構成より精度が高い。OCR は打ち消し線の上下にある文字を個別にテキストとして抽出するため、有効な値の判断が困難になるためである。	打ち消し線を含む帳票の処理精度改善	4.3.4
③プロンプト設計	LLM へ与えるプロンプトの設計次第で、精度はさらに改善できる。特に選択式項目では Few-shot が有効である。	LLM への指示文最適化による精度改善	4.3.5
④ ファインチューニング	プロンプト改善や前処理改善で解消できない精度課題に対し、モデル自体を帳票ドメインに適応させるファインチューニングにより、標準モデルを上回る精度を達成できるか。	VLM の追加学習による精度改善の可能性検証	4.3.6

これらの仮説を検証するにあたり、精度改善のボトルネックが「プロンプト改善」や「前処理改善」で解消可能か、あるいは「モデル自体の学習（ファインチューニング）」に依存するのかを切り分けることを検証設計の基本方針とした。

4.3.2　評価方法と計測指標

対象データの選定

評価対象はサブ検証の内容によって適切なデータが異なるため、サブ検証項目ごとに選定する。選定の基準は、第 1 に出力スキーマ数が多く複雑な読み取りが必要なため、モデル間の差異が顕在化しやすいこと。第 2 に、丸囲み・手書き・打ち消し線といった視覚的解釈を要する要素と、自由記述による文章抽出が混在しており、比較しやすい帳票特性を持つデータを取り扱う。

評価軸の設計

以下の 5 軸を共通評価軸として設定し、同一条件で結果をまとめた。

表 4.2　評価軸一覧

評価軸	内容

精度	全体の正答率に加え、文章抜き出し系と選択式・視覚的要素を分けて確認
処理速度	推論にかかる実時間（分単位）
コスト	API コスト・インフラコスト（今回は定性的な比較に留める）
運用上の扱いやすさ	再実行単位・エラー切り分け・安定性の観点
拡張性	スループット・並列実行・モデル追加の容易性

評価指標：全カラム平均精度

精度評価の主指標として、全カラム平均精度（クレンジング後）を用いた。各ファイルに対して定義されたすべての抽出カラムについて、モデルが出力した値と正解ラベルを比較し、一致したカラムの割合を算出する。これをファイル間で平均したものを全カラム平均精度とした。「クレンジング後」とは、抽出値の表記揺れ（全角・半角の違い、余分な空白等）を正規化した後に比較を行うことを指す。

表 4.3　評価カテゴリ一覧

評価カテゴリ	説明
文章抜き出し精度	自由記述欄から文章を正確に抽出できるかを評価
丸囲み精度	選択式（丸囲み・チェックボックス等）の選択結果を正確に抽出できるかを評価
全カラム平均精度	上記を含む全カラムの精度の平均値

4.3.3　サブ検証①：OCR モデルの横並び評価

仮説

仮説 1（OCR 構成）を検証する。VLM 単体より、OCR と LLM を組み合わせた構成の方が安定した精度を発揮するか。

検討した選択肢

以下の 4 つのモデルパターンを同一データ・同一スキーマで評価した。マネージド LLM として、現行構成で実績のある Claude 3.5 Sonnet を精度の上限（ベースライン）、低コスト・高スループットの Claude 3.5 Haiku を対照として選定した。OSS 系 VLM として、画像入力により視覚的要素を直接扱える Qwen2.5-VL（72B）、および日本語業務文書への最適化が期待できるリコー系モデルを候補とした。

LLaMA 3.2 Vision は日本語帳票文字の適性に課題があり除外した。Gemini 2.0 Pro は日本リージョンがなく不使用とした。自動車事故報告書別記様式（第 3 条関係）を対象とした。

表 4.4　比較モデル一覧

モデル名	種別	パラメータ数	トークン制限	備考

Claude 3.5 Sonnet	マネージド LLM	非公開	1M	高精度だがコスト高
Qwen2.5-VL 3B / 72B	OSS VLM	3B / 72B	約 1M	LoRA 可。 画像＋テキスト対応
リコーLLM	商用日本語 LLM	70B	—	日本語業務文書向け

結果

表 4.5　モデルパターン別評価結果

モデルパターン	全カラム平均精度	文章抜き出し精度	丸囲み精度	推論処理速度
Azure OCR + Claude 3.5 Sonnet（LLM） ※現行 Veda 構成	0.7764 （77.6%）	0.8307	0.7008	16.45 分
Claude 3.5 Sonnet （VLM 単体）	0.6569 （65.6%）	0.5640	0.6895	4.40 分
Qwen2.5-VL-72B （VLM 単体）	0.6625 （66.2%）	0.6018	0.6548	13.00 分
リコーVLM （VLM 単体）	0.5912 （59.1%）	0.5588	0.6444	231.00 分

精度面では Azure OCR + Claude 3.5 Sonnet（LLM）が最も良好であり、文章抜き出し精度（0.8307）と丸囲み精度（0.7008）の両面で相対的に高い水準を示した。VLM 単体は処理速度で優位性があるものの、全体精度で OCR＋LLM 構成に対して 10 ポイント以上の差が生じた。

結論

仮説 1 は支持された。今年度の標準構成は OCR＋LLM（Azure OCR + Claude Sonnet）を採用した。VLM 単体は、丸囲み・打ち消し線など画像解釈が必要な箇所に限定して併用する方針とした。

4.3.4　サブ検証②：打ち消し線検知の高度化

仮説

仮説 2 の延長として、打ち消し線の有無を構造化処理の前段で検知する分類モデルを導入し、打ち消し線がないページは OCR＋LLM で処理、打ち消し線があるページは VLM で処理するという二段判定により、全体の構造化精度が改善できると仮説を立てた。そのため、そもそも処理対象ページに打ち消し線が含まれているかどうか自体を判定する最適な手段を調査した。

検討した選択肢

表 4.6　打ち消し線検知のアプローチ一覧

アプローチ	概要
OSS モデルのセルフホスティング（SigLIP 2）	GPU 搭載のサーバー環境上でセルフホストし、画像前処理や学習データ設計を含めて細かくチューニング
マネージドサービス（Azure Custom Vision）	学習・推論基盤の運用をマネージド側に委ねる方式

結果①：SigLIP 2 による検証

SigLIP 2（Google が開発した視覚-言語モデル。画像とテキストを対応付ける事前学習を行っており、本検証では貨物自動車運送事業報告規則に基づく事業実績報告書（4 号様式：貨物自動車運送事業事業実績報告書）を対象に、打ち消し線の有無を二値分類する目的でファインチューニングして利用した）を GPU 搭載のサーバー環境上でファインチューニングした結果、F1 スコア（適合率と再現率のバランスを示す総合精度指標。100%が完璧な精度 ）93.38%を達成した。

表 4.7　SigLIP 2 による打ち消し線検知精度

指標	精度(%)
Precision （適合率：「ある」と判定したもののうち実際にそうだった割合）	93.36
Recall	93.42
F1 スコア	93.38

しかし、運用面で 2 つの課題が判明した。第 1 に、処理時間が約 1,700 秒と長く全体処理のボトルネックとなった。第 2 に、事前学習のデータではない、自動車事故報告書に適用したところ F1 スコアが 68.98%まで低下し、汎用性に課題があることが示された。

結果②：Azure Custom Vision への切り替え

SigLIP 2 の課題を踏まえ、Azure Custom Vision に切り替えた。打ち消し線のある 1,871 ファイルを対象に、学習 8 割・テスト 2 割で分割してモデルを作成した。

表 4.8　Azure Custom Vision による打ち消し線検知精度

指標	精度(%)
Precision	85.3
Recall	84.1
F1 スコア	84.69

表 14.9　混同行列（テスト 100 ファイル：打ち消し線あり 50、なし 50）

	予測:打ち消し線あり	予測:打ち消し線なし
実際:打ち消し線あり	TP = 14	FN = 36
実際:打ち消し線なし	FP = 2	TN = 48

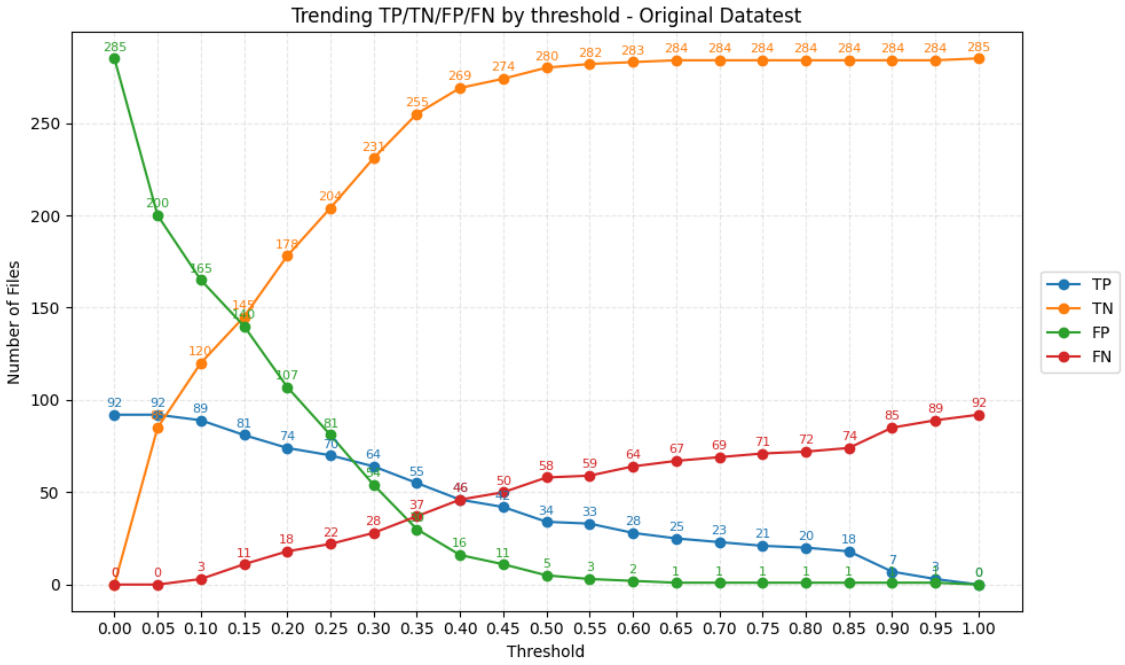


図 21　しきい値別の TP/TN/FP/FN トレーディンググラフ結果

採用アーキテクチャ（二段判定）

以上の知見を踏まえ、以下の二段判定アーキテクチャを採用した。①文書全体に対して打ち消し線の有無を Azure Custom Vision で検知する。②打ち消し線が検知されたページは VLM で処理し、検知されなかったページは OCR＋LLM で処理する。③打ち消し線が含まれるフィールドについてのみ、VLM の抽出結果で最終値を上書きする。Azure Custom Vision のモデルでは打ち消し線検知の推定確率が出力、指定したしきい値を超えた場合に打ち消し線ありと判定する。「見逃し（FN）」がないようにするため、FN が最小となる 0.05 をしきい値として定め、疑わしいケースは VLM で二段確認する運用とした。

これにより、訂正がない大部分の領域では OCR＋LLM の処理効率と安定性を活用しつつ、打ち消し線が存在する箇所に限って VLM の視覚的な判定能力を適用する構成を実現した。

結論

打ち消し線検知は SigLIP 2 で高精度を確認できたが、処理時間・汎用性の課題から Azure Custom Vision に切り替えた。しきい値を低めに設定し FN を抑制する二段判定アーキテクチャにより、OCR＋LLM と VLM の役割分担を最適化した。

4.3.5　サブ検証③：プロンプトエンジニアリングの検証

仮説

LLM モデルを変更せずとも、判断基準や事例を用意するなどより高度なプロンプトエンジニアリングを用いることで、構造化精度の改善を図ることができるのではないか。

手法

以下の 5 つのプロンプトエンジニアリング手法を段階的に導入し、各手法が精度にどのような変化をもたらすかを自動車事故報告書別記様式（第 3 条関係）を対象に検証した。

表 4.10　実施したプロンプトエンジニアリング手法一覧

手法	説明	プロンプト例
プロンプトエンジニアリングなし	事前の例示なしで指示のみで実行	-
Few-shot	具体的な入出力例を少数提示してから実行させる手法	以下に、特に間違いやすい項目に関する、他のドキュメントからの正しい抽出例を示します。フォーマットと内容（特に 0 と null の使い分け）を厳密に参考にしてください。 例: チェックボックスが空欄の場合（0 を使用） 入力ドキュメント: (事故の種類セクションにチェックがない状態を想定）正しい JSON 出力（抜粋）: JSON { "事故の種類_区分_車両故障": 0, "事故の種類_発生の順_衝突": 0, "事故の種類_発生の順_車両故障": 0 }
Step Back	文書全体の構造を理解させてから詳細に取り組む手法	ドキュメント全体を迅速にレビューし、これが「自動車事故報告書」の定型フォーマットであることを認識します。主要なセクション（ヘッダー、事故の種類、自由記述欄など）の位置を大まかに把握します。

Chain-of-Thought（CoT）	段階的な思考プロセスを明示的に促す手法	出力スキーマの各項目を一つずつ、以下の指示に従って論理的に特定し、抽出します。 ヘッダー情報の特定: 文書上部の表から「氏名又は名称」「日時」「場所」などの基本情報を探します。 指示: どの選択肢がマークされているか（丸で囲まれている、チェックされているなど）を特定します。次に、その選択肢のテキストラベル部分のみを正確に抽出します。例えば、「道路(① 高速自動車国道)」という記載で①が選択されている場合、"高速自動車国道"という文字列だけを抽出します。
役割付与（bothinst）	専門家としての役割を付与しタスクへの取り組み方を調整する手法	あなたは、報告書のような複雑で半構造化された文書から、構造化情報を 100%の精度で抽出することを専門とする、極めて厳格なデータアナリストです。

プロンプト設計の要点は以下のとおりである。

役割付与

手書き修正が頻繁に含まれる行政文書の解釈を専門とするデータアナリストとして振る舞うよう指示した。

Step Back + CoT

①文書構造と全体像の把握→②修正箇所の特定とフィルタリング→③JSON 構築と検証、の 3 段階の思考プロセスを設定した。

絶対ルールの明文化

打ち消し線が引かれた旧情報の完全破棄（絶対ルール 1）、選択式項目の厳格な解釈（絶対ルール 2）を明示し、違反をタスク失敗と定義した。

Few-shot

正解例・NG 例を具体的に提示した（例：{"item": "旧情報 新情報"}は NG、{"item": "新情報"}が OK）。

結果

表 4.11　プロンプトエンジニアリングの評価結果

指標	プロンプトエンジニアリング なし	Few-shot	Step Back + CoT	すべて併用
全カラム正答率	0.6350	0.6423	0.6383	0.6429

丸囲み正解率（役割付与あり/なし）	0.6411 / 0.7773	0.7201 / 0.7727	0.7205 / 0.7788	0.7795
文章抜き出し正解率（役割付与あり/なし）	0.3232 / 0.5070	0.3406 / 0.5456	0.2132 / 0.5193	0.5395

実際に使用したプロンプト

以下は、Zero-shot・Few-shot・Step Back・CoT・役割付与のすべてを組み合わせた検証で実際に使用したプロンプト全文の日本語訳である。実際のプロンプトの言語は LLM モデルが最も学習している言語と考えられる英語となっている。

データ構造化(LLM)の場合のプロンプト（日本語訳）

```
概要
あなたは、文書から情報を正確に抽出し、指定されたキーを持つ JSON を返すエンジンです。
コアルール
基本要件
出力は、指定されたカラム名と完全に一致するキーを持つ「単一の JSON オブジェクト」のみとする。出力にコードフェンス（```）を含めない（JSON のみを出力すること）
数値の表記ルール
数値は半角で出力する
カンマは付けない
単位換算はしない（例：千円単位の金額は千円単位のまま）
△ / ▲ 記号はマイナス記号「-」に置換し、数値として出力する
パーセンテージは小数点を含む数値として出力する（%記号は付けない）
文字列の扱い
住所・所在地：都道府県名または市区町村名から抽出する（郵便番号は除外する）　JSON 文字列：JSON 内部のダブルクォーテーション（"）は必ず \" としてエスケープする　空の値：空文字列は使わず、必ず null を使用する（空文字は禁止）
日付・時刻の形式
文書に記載された形式に従って日付・時刻を抽出する
時刻が記載されている場合は、時刻も含めて抽出する
テーブル型カラムに関する特別指示
テーブル形式：Markdown（デフォルト）
Markdown 表を「文字列配列」として返す
1 つの表は「1 つの文字列」として返す（行ごとに分割しない）
複数の表がある場合は、配列の複数要素として返す
```

【最重要】行・列・データを省略せず、完全に抽出すること
OCR 検出結果＋画像解析を使い、最も適切な表を選択して再構成すること
テーブル形式：JSON
flat / auto：フラットな行指向の JSON オブジェクトとして返す
各行をフラットな辞書として表現する
hierarchical：階層構造を保持した JSON オブジェクトとして返す
複雑な多階層テーブル構造を保持する
実装例（テーブル用プロンプト文の書き方）
テーブル型カラム向けの指示文を組み立てる場合は、以下の例を参考にすること。
Markdown: "列『{col.name}』（table 型）：マークダウン表の文字列配列で返す。1 つの表は 1 つの文字列として返すこと（行ごとに分割しない）。複数の表がある場合は配列の複数要素として返す。表の全ての行・列・データを省略せずに完全に抽出すること。OCR で検出された表と画像を使って、最も適切な表を選択・再構成すること。"
JSON（Flat）: "列『{col.name}』（table 型）：フラット行指向型の JSON オブジェクトで返す。各行をフラットな辞書として表現すること。"
JSON（Hierarchical）: "列『{col.name}』（table 型）：階層構造保持型の JSON オブジェクトで返す。複雑な多階層テーブルの構造を保持すること。"
テンプレート変数
実装時は以下の変数を置換すること。
{col.name}: スキーマ上のカラム名
{col.table_format}: "markdown"（デフォルト）または "json"
{col.table_style}: "flat" / "hierarchical" / "auto"
補足
このテンプレートは、スキーマ上のカラムに応じて動的に組み立てられることを想定する
table 型カラムが存在する場合のみ、テーブル向けの追加指示を付与する
指示はすべて累積的に適用される（後から書いたものが無効化するとは限らない

訂正線検知・修正(VLM)の場合のプロンプト（日本語訳）

高精度データ抽出のための汎用指示書
1. あなたの役割
あなたは、OCR でテキスト化された、手書きによる修正（丸囲み、訂正線、追記など）が頻繁に含まれる複雑なビジネス文書や行政文書の解釈を専門とする、超熟練のデータアナリストです。あなたの最優先事項は、手書き修正の意図を正確に読み取り、ノイズ（訂正前の情報や非選択項目）を完全に排除し、100%クリーンで正確な構造化データ（JSON 形式）を生成することです。
2. タスクの定義
提供されたドキュメントイメージから、以下の思考ステップと絶対ルールに基づき、手書きの修正や曖昧な記号を厳密に解釈・フィルタリングし、最終的に有効な情報のみを反映した単一の JSON オブジェクトを出力してください。

以下の思考ステップと絶対ルールは、あなたのタスクにおける唯一の行動規範です。特に【絶対ルール 1】と【絶対ルール 2】の遵守は最優先事項であり、これらのルールに一度でも違反した出力は、タスク全体の失敗とみなされます。いかなる場合も自己判断でルールを曲解せず、機械的かつ厳密に適用してください。

注意: この画像には訂正線で消された古い情報と、その付近に手書きで追記された新しい情報が存在します。
あなたのタスクは、まず古い情報を特定し、それを完全に破棄することです。次ステップとして、新しい情報のみを抽出し、それを最終的な値としてください。
3. 思考ステップと厳密な解釈ルール
注意: この画像には訂正線で消された古い情報と、その付近に手書きで追記された新しい情報が存在します。
あなたのタスクは、まず古い情報を特定し、それを完全に破棄することです。次ステップとして、新しい情報のみを抽出し、それを最終的な値としてください。
ステップ 1: 文書構造と全体像の把握（Step Back）
まず、文書全体をスキャンし、これがどのような報告書（例：事業実績報告書、在庫棚卸表など）であるかを理解します。表のレイアウト、ヘッダー情報（会社名、住所、報告日など）、明細項目、備考欄の配置を把握します。この段階では、訂正線で消された文字や記号も含め、目に見えるすべての情報を内部的に認識します。

中間ステップ：項目の種類判定
各項目を抽出する前に、それが「自由記述項目」か「選択式項目」かを以下の基準で判定します。

「選択式項目」とみなす基準： 複数の項目がリスト形式で並んでおり、かつ各項目の先頭に選択マーカ（〇, □, ✓など）を記入するための記号、あるいはそのための空白が存在する場合。
項目が「選択式項目」と判定された場合は、後述の【絶対ルール 2】を他のどのルールよりも優先して厳格に適用してください。
ステップ 2: 修正箇所の特定とフィルタリング（最重要プロセス）
次に、ステップ 1 で認識した情報に対し、以下の【絶対ルール】を適用して、有効な情報と無効な情報
（破棄すべき情報）を厳密に仕分けします。破棄すべき情報は絶対に破棄し、null としてください。
以下は特に大切なルールです。破棄すべき情報は絶対に null にしてください。
【絶対ルール 1：訂正・修正箇所の解釈】
訂正線の認識: 1 本線、または 2 本線、または 3 本線、または塗りつぶしによって明確に消去されている
テキストや数値（以下、「旧情報」）を特定します。

修正情報の探索: 「旧情報」の直上、直下、またはすぐ隣に手書きで追記された新しいテキストや数値
（以下、「新情報」）を探します。
情報の置換:
「新情報」が見つかった場合、「旧情報」は完全に破棄し、「新情報」を唯一の正しい値として採用します。
出力には「新情報」のみを含めてください。
　NG 例（誤った処理）: {"item": "旧情報 新情報"} や {"item": "旧情報"}
　OK 例（正しい処理）: {"item": "新情報"}
関連不明な追記: もし手書きの追記が、どの項目を修正したものか文脈上判断できない場合は、その追記を無視するか、別途 remarks フィールドに格納します。
【ペナルティ】
このルールに違反し、訂正線が引かれた旧情報を少しでも出力に含めた場合、あなたのタスクは失敗です。細心の注意を払ってください。
【絶対ルール 2：選択式項目の厳格な解釈】
これは「選択式項目」と判定された項目にのみ適用される最優先ルールです。
抽出の絶対条件:
有効な選択マーカー（完全に閉じられた円〇、チェックマーク ✓ など）が、項目テキストに物理的に
隣接している場合「のみ」、そのテキストを抽出します。
まず、リストや表形式で複数の項目が並んでいる場合、それらを「選択肢」として認識します。
有効な選択マーカーの定義:
　塗りつぶされた記号: ●、■ など
　完全に閉じられた円: 〇
　チェックマーク: ✓
無視する非選択マーカーの定義:
　二重丸・穴あき丸: ◎
　不完全な円: C 字型など
　空の四角・括弧: □、[ ]、( )
　小さい黒丸: ・
完全破棄の原則:
有効な選択マーカーが隣接していないテキストは、たとえそれがリストの一部であっても、選択されなかったノイズとみなし、完全に出力から除外します。いかなる理由があっても出力に含めてはなりません。
【ペナルティ】
このルールに違反し、訂正線が引かれた旧情報を少しでも出力に含めた場合、あなたのタスクは失敗です。
以下は特に大切なルールです。破棄すべき情報は絶対に null にしてください。

【絶対ルール 3：訂正・修正箇所の無視】
訂正線の認識: 1 本線、または 2 本線、または 3 本線、または塗りつぶしによって消去されているテキストや数値（以下、「旧情報」）を特定します。（消去されていると明確には分からなくても、周辺のテキストとは違う背景情報となっている部分は旧情報と見做します。）
修正情報の探索: 「旧情報」の直上、直下、またはすぐ隣に手書きで追記された新しいテキストや数値
（以下、「新情報」）を探します。
情報の置換:
「旧情報」は完全に破棄し、null としてください。「新情報」を唯一の正しい値として採用します。
出力には「新情報」のみを含めてください。
　NG 例（誤った処理）: {"item": "旧情報 新情報"} や {"item": "旧情報"}
　OK 例（正しい処理）: {"item": "新情報"}
関連不明な追記: もし手書きの追記が、どの項目を修正したものか文脈上判断できない場合は、その追記を無視するか、別途 remarks フィールドに格納します。
【ペナルティ】
このルールに違反し、訂正線が引かれた旧情報を少しでも出力に含めた場合、あなたのタスクは失敗です。
細心の注意を払ってください。
ステップ 3: 最終 JSON の構築と検証
構築: ステップ 2 でフィルタリングされた有効な情報のみを使用して、JSON オブジェクトを構築します。
なお、たとえ同じ内容が記載されていたとしても、違う場所に記載されていた内容は全て抽出します。
（例：九州という見出しの情報と全国という見出しの項目の内容が同じであっても、どちらも出力に含めてください）
ヘッダー情報: 文書全体に共通する情報（会社名、報告年度など）は、ルートレベルにキーとして配置します。添付のスキーマ指定ファイルを参照してください。
明細データ: 表形式のデータは、各行を 1 つのオブジェクトとする配列（Array）として構造化します。列見出しをキーとして使用してください。
日付の正規化: 文書内の日付（例：令和 5 年 6 月 19 日）は、西暦の ISO 8601 形式（YYYY-MM-DD）に正規化します（例：2023-06-19）。元号は西暦に変換してください。
検証: 生成した JSON を再度レビューし、【絶対ルール 1】および【絶対ルール 2】に違反するデータが含まれていないか（例：訂正線が引かれた旧情報が残っていないか、非選択の◎項目が含まれていないか）を最終確認します。
追加検証：選択項目のチェック
「選択式項目」として処理したフィールドについて、生成された JSON 配列の要素数が、元の画像上で手書きされた有効な選択マーカー（〇や✓）の数と完全に一致しているかを確認してください。もし一致していなければ、あなたの解釈が誤っている証拠です。ステップ 2 のルー

ルに立ち返り、解釈をやり直してください。
4. 出力形式
思考プロセスや説明文は一切含めず、最終的な JSON オブジェクトのみを出力してください。
有効な情報が存在しない項目は、空文字（""）ではなく null を使用してください。

結論

選択式項目（丸囲み等）では Few-shot が有効であり、仮説 3 は部分的に支持された。一方、文章抜き出しの改善幅は限定的であり、OCR 誤りやレイアウト揺れの影響が支配的であることが確認された。最も精度に寄与したのは、手法の追加よりも「出力ルールの言語化・固定（0/1/null・ラベル抽出・改行結合等）」であった。この結果を踏まえ、プロンプト最適化手法をすべてを全ユースケースへ標準導入する方針とした。

4.3.6　サブ検証④：ファインチューニングの検討

仮説

プロンプト改善や前処理改善で解消できない精度課題に対し、モデル自体を帳票ドメインに適応させるファインチューニングにより、標準モデルを上回る精度を達成できるか。

手法

ファインチューニングの対象モデルとして、Qwen2.5-VL（3B パラメータ）を選定した。なお、B とは billion（億）の略であり、LLM モデルのパラメーター数を指す。一般的に、パラメーター数が多いほどモデルの性能は高くなるが、必要となる計算資源も大きくなる。Qwen2.5-VL はテキスト・画像の両方を入力として扱えるオープンソース VLM であり、LoRA（低ランク適応：モデル本体を変更せず低ランク行列のアダプターのみを学習する手法であり、LLM のファインチューニングとして一般的に広く用いられる方法）によるパラメータ効率の良い追加学習が可能である。
なお、72B モデルへのファインチューニングは、学習に必要な GPU リソース（p4d.24xlarge 相当）のコストが大きく、本年度の検証スコープでは実施していない。
自動車事故報告書別記様式（第 3 条関係）を対象とし、検証条件は、学習 252 件・評価 63 件（画像）とした。インフラは SageMaker g5.xlarge（$0.97/h）（Sagemaker: データ前処理、モデル構築、学習、デプロイまでを統合して提供する AWS のフルマネージド型機械学習サービス）を用いた。推論時は EC2 によるホスティングが想定される。

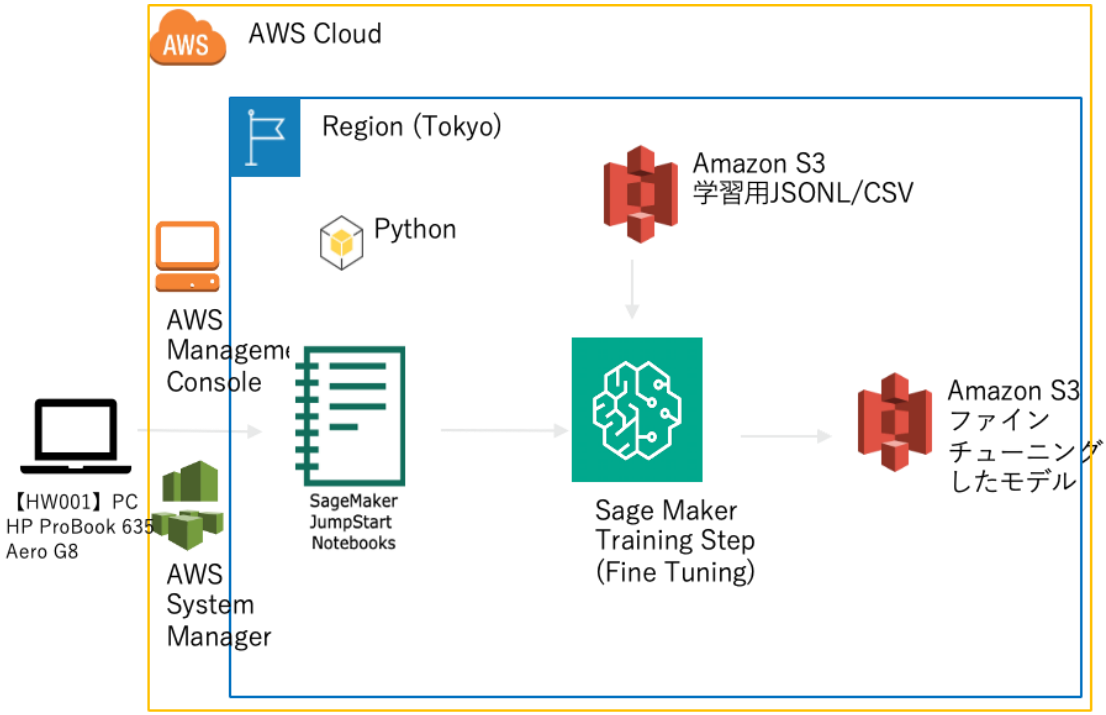


図 22 AWS をインフラとした場合のシステム構成図（ファインチューニング時）

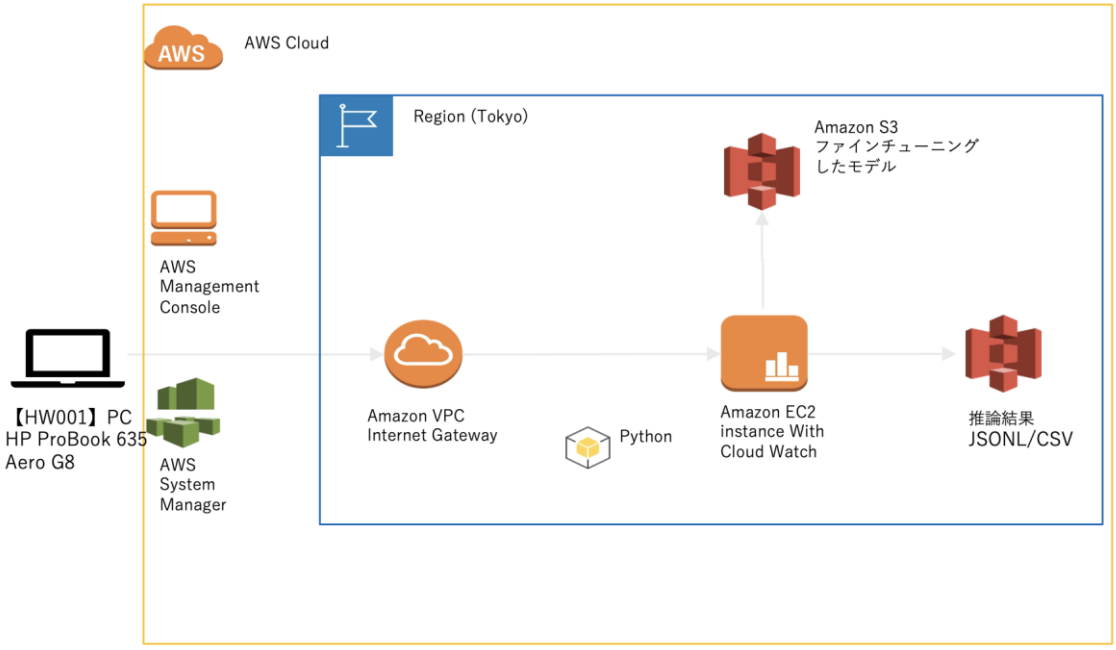


図 23 AWS をインフラとした場合のシステム構成図（ファインチューニングモデルを用いた推論時）

結果

表 4.12　ファインチューニング（FT）検証結果

構成	全カラム平均精度	学習時間	推論速度	学習コスト	推論コスト	学習件数	推論件数

Qwen2.5-VL 3B（FT なし）	0.0750（7.5%）	—	2.1 分	—	約 2.0 ドル	—	63
Qwen2.5-VL 3B（FT あり・LoRA）	0.4333（43.3%）	305 分	2.1 分	約 4.93 ドル	約 0.03 ドル	252	63
参考：Qwen2.5-VL 72B（FT なし）	0.5411（54.1%）	—	20.7 分	—	約 13.0 ドル	—	20
参考：Azure OCR + Claude 3.5 Sonnet	0.7764（77.6%）	—	16.2 分	—	約 16.45 ドル	—	20

ファインチューニングにより、3B モデルは 7.5%から 43.3%へ大幅に改善された。ただし、OCR＋Sonnet の 77.6%には大幅に及ばなかった。72B モデルはファインチューニングなしでも 54.1%を実現しており、3B への FT 適用（43.3%）を上回った。

結論

ファインチューニングはモデル精度を向上させる効果が確認されたが、現時点では OCR＋LLM 構成の精度に及ばない。加えて OSS モデルの自前ホスティングには GPU 搭載サーバー環境の維持・スケーリング・障害対応などの継続的なコストが必要となる。今年度は本番導入を見送り、将来の OSS 活用選択肢として継続検討とした。

4.3.7　モデル自動選定

課題

4.3.3〜4.3.7 のサブ検証を通じて「OCR＋LLM（Azure OCR + Claude Sonnet）」を標準構成として採用したが、高性能モデル（Sonnet）の常時利用はコスト増大につながる。一方で低コストモデル（Haiku）への一律切り替えは、丸囲みを含む資料等で品質低下を招く。入力データ特性に応じてモデルを自動的に振り分け、コスト最適化と品質維持を両立する仕組みが必要である。

手法

以下の 2 軸で自動振り分けロジックを設計した。

① 出力一致率（Similarity）

Haiku と Sonnet の出力を比較し、一定の一致率を満たす場合は Haiku を採用する。

② 丸囲み（図形）有無の検知

丸囲みが含まれる場合は Sonnet を優先する（特徴トリガー）。
検証には丸囲み（図形）が含まれる 3 つのデータで検証した。

結果

表 4.13　しきい値別のモデル自動選定結果
（検証データ①倉庫業登録申請書 15 件）

しきい値	Haiku が選定された件数（割合）	Sonnet が選定された件数（割合）
90%	13（86.7%）	2（13.3%）
95%	11（73.3%）	4（26.7%）
99%	6（40.0%）	9（60.0%）

表 4.14　しきい値別のモデル自動選定結果
（検証データ② 国会答弁データ 20 件）

しきい値	Haiku が選定された件数（割合）	Sonnet が選定された件数（割合）
90%	19（95.0%）	1（5.0%）
95%	15（75.0%）	5（25.0%）
99%	4（20.0%）	16（80.0%）

表 4.15　しきい値別のモデル自動選定結果
（検証データ③ 建設工事施工統計調査 7 件）

しきい値	Haiku が選定された件数（割合）	Sonnet が選定された件数（割合）
90%	7（100%）	0（0%）
95%	7（100%）	0（0%）
99%	2（28.6%）	5（71.4%）

データの特性によって効き方が大きく異なることが確認された。**検証データ①**は一致率が低いファイルが混在し Similarity のみでは分類が成立しにくかった一方、丸囲みトリガーが有効であった。**検証データ②**では Similarity 分布が高くしきい値による段階的振り分けが機能しやすかった。**検証データ③**では丸囲みを含む資料が多く、丸囲み検知がモデル選択を大きく左右した。

運用上のしきい値は 99%を採用する。90%や 95%では差分が残るケースまで Haiku に倒れるリスクがあり、100%では全件 Sonnet となりコスト最適化の余地がなくなるためである。

結論

モデル自動選定は「丸囲み等の特徴トリガー優先＋一致率 99%しきい値による段階判定」のハイブリッド方式を採用した。残課題として、データ特性差を踏まえたしきい値の最適化、丸囲み以外の追加トリガー（手書き、打ち消し線、表構造複雑度等）の検討が必要である。

4.3.9　小括

課題 A（構造化精度の向上）に対する 5 つのサブ検証およびモデル自動選定の結論を以下に整理する。

本年度のデータ構造化の標準構成は、2024 年度と同様に OCR＋LLM（Azure OCR + Claude Sonnet）を基軸とし、打ち消し線や丸囲み等の画像認識が必要な項目に対しては VLM 併用で補完する方針とした。

OCR モデル選定（4.3.3）

Azure OCR + Claude Sonnet（LLM）が全カラム平均精度 77.6%で最も安定し、標準構成として採用した。VLM 単体は精度面で 10 ポイント以上の差があり、画像解釈が必要な箇所に限定して併用する。

打ち消し線検知（4.3.4）

Azure Custom Vision による二段判定アーキテクチャを採用した。打ち消し線ありページのみ VLM で再判定し、該当箇所だけを上書きすることで再処理範囲を最小化した。

プロンプトエンジニアリング（4.3.5）

手法追加よりも出力ルールの明文化・固定が精度に最も寄与した。データ構築モジュールへ標準導入する方針とした。

ファインチューニング（4.3.6）

精度向上効果は確認できたが、OCR＋LLM 構成の 77.6%には及ばず、今年度は本番導入を見送り継続検討とした。

Excel 図形（4.3.7）

PDF 変換＋HTML 変換のハイブリッド方式を採用した。密集ケースの誤解釈は継続改善とした。

モデル自動選定（4.3.8）

特徴トリガー優先＋一致率 99%しきい値のハイブリッド方式を採用した。データ特性差に応じた最適化は継続課題である。

4.4　課題 B：大規模ドキュメントの処理

本節では、課題 B（大規模ドキュメントの処理）に対する検証を「仮説→手法→計測方法→検討した選択肢→結果→結論」の順で整理する。

4.4.1　仮説

100〜200 ページに及ぶ行政文書をそのまま LLM に投入すると、入力トークン上限を超過し処理が失敗する。4.2 節で述べたとおり、機械的にチャンク分割しても文脈が分断されて抽出精度が低下する問題が確認されている。

この課題に対し、以下の仮説を設定した。LLM のトークン上限を考慮し、入力トークン数を事前に推定したうえで処理単位を自動分割すれば、トークン超過による処理失敗を防止しつつ、分割単位内で文脈を保持した情報抽出が可能になる。

4.4.2　手法

OCR によるテキスト抽出後、LLM への入力前に以下の 2 指標を用いてトークン数を推定し、LLM コンテキスト上限（現在 180,000 トークン）との比較を行う自動分割機構を導入した。

① 非構造ファイルのページ数情報およびテキスト量

OCR 処理後に得られるページごとのテキスト量から、ファイル全体の推定トークン数を算出する。

② LLM コンテキスト上限との比較

推定トークン数がコンテキスト上限（180,000 トークン）を超える見込みがある場合、最大 20 ページ/バッチで複数バッチに自動分割し、バッチごとにデータ構造化処理を実行する。

4.4.3　計測方法

自動分割の有効性は、100 ページ以上の PDF ファイルを対象に、分割なし（トークン超過でエラー）と分割あり（バッチ処理）でのデータ構造化処理の成否を比較することで評価した。検証には、自治体や事業者が事業計画と必要予算を国へ伝え、予算枠を確保するための要望書データを用いた。データ選定の理由は 100 ページを超える PDF データである上、抽出したい項目が記載されているページが合計で 10 ページ程度のため、機能の検証として適しているためである。

4.4.4　検討した選択肢

大規模文書の分割戦略として、以下の選択肢を検討した。

表 4.16　大規模文書の分割戦略の選択肢

選択肢	概要	利点	課題
固定ページ数による分割（採用）	最大 20 ページ/バッチで機械的に分割	実装がシンプル。トークン上限の超過を確実に防止できる	分割境界がページ単位のため、文脈がページをまたぐケースで抽出精度が低下する可能性がある
重要ページ抽出	キーワードをもとに、出力スキーマとの関連性の高いページのみを抽出	出力スキーマに応じた抽出箇所の絞り込みが可能	ユーザーに対する、抽出失敗した場合の見落とし防止や抽出根拠の提示が困難

今年度は、実装のシンプルさとトークン超過防止の確実性を優先し、固定ページ数による分割を採用した。

4.4.5　結果

データ自動分割の導入により、100 ページ以上の PDF に対してもトークン上限を超過することなくデータ構造化処理を完了できるようになった。最大 20 ページ単位での処理となるため、各バッチ内では資料上の文脈を踏まえた情報抽出が可能であることを確認した。
一方で、バッチ境界をまたぐ文脈の分断（例：20 ページ目と 21 ページ目に意味的なつながりがある場合）については、今回の検証対象では該当するケースが確認されなかった。ただし、論理的には分割境界で文脈が失われるリスクが存在するため、今後の実運用で顕在化する可能性がある。

4.4.6　結論

固定ページ数（最大 20 ページ/バッチ）による自動分割により、大規模文書のトークン超過問題は解消された。仮説は支持された。
ただし、現在の固定ページ数方式はあくまで暫定的な対処であり、文書の意味的構造を考慮していない。将来的には、章・セクション単位の分割やオーバーラップ付き分割など、文脈保持を考慮した分割戦略への発展が求められる。また、分割境界をまたぐ文脈の分断が実運用で問題となるケースが確認された場合には、優先的に対応する必要がある。

4.5　課題 C：スケーラビリティの改善

本節では、課題 C（スケーラビリティの改善）に対する検証を「仮説→手法→計測方法→検討した選択肢→結果→結論」の順で整理する。

4.5.1　仮説

4.2 節で述べたとおり、2024 年度時点ではデータ構造化処理は 1 ページあたり平均 45 秒を要し、仮に並列処理を行わない場合には 100 ページの文書に換算すると文書 1 件で約 75 分、1 日に換算すると数十件の文書を処理する程度に留まる。並列処理なしでは実務に耐えることは現実的ではない。
この課題に対し、スループットのボトルネックは主に以下の 2 点にあり、これらの制約を緩和しつつ並列処理を最適化することで、単位時間あたりの処理ファイル数を大幅に改善できるという仮説を設定した。

ボトルネック①：AWS Bedrock の API レートリミット

AWS Bedrock に対する API リクエストは、AWS アカウントに紐づいたレートリミット（単位時間あたりの最大リクエスト数・最大トークン数）により制限される。この上限を超えたリクエストはスロットリング（処理が制限・遅延させられること）され、再試行が必要になる。レ

ートリミットは AWS アカウント単位で管理されており、デフォルト値のままでは大量の並列処理が困難であった。

さらに、2024 年度において一括処理において Claude 3.5 Sonnet の 1 分間リクエスト数の制限があったために、制限を超えると Haiku 処理に切り替わる実装をとっていた。しかし、Sonnet 処理比率の低下が精度低下につながることが確認されたため、本切替方式は廃止した。この問題への対処として、複数リージョン・複数モデルの分散活用が検討された。

ボトルネック②：LLM の処理能力

データ構造化処理（OCR・LLM 呼び出し・後処理）は 2024 年度に貨物自動車運送事業報告規則に基づく事業実績報告書（4 号様式：貨物自動車運送事業事業実績報告書）を対象とした際に、1 ページあたり平均 44.5 秒程度（OCR 処理：9.5 秒/ページ、データ構造化：12 秒/ページ、不正検知：23 秒/ページの合計）を要した。複数ファイルをシリアルに処理する構成では、ファイル数に比例して合計処理時間が増大する。

4.5.2　手法

上記 2 つのボトルネックに対し、以下の 4 つの対策を組み合わせて実施した。

レートリミットの引き上げ

AWS 社と連携し、本プロジェクトの AWS アカウントに対する Bedrock の API レートリミット（1 分あたりのリクエスト数・トークン数上限）の引き上げを申請・実施した。以下の表は、AWS Bedorock API のレートリミット引き上げ前後における 1 分あたりのトークン数と 1 秒あたりのリクエスト数の比較表である。

表 4.17　AWS Bedrock レートリミット引き上げ前後の比較

項目	引き上げ前	引き上げ後
トークン数/分	400,000	5,000,000
リクエスト数/秒	200	1,000

加えて、スループット制限に起因するエラーが発生した場合、自動リトライ処理を実装し、安定してリクエストを処理できるように対応した。

また、1 リクエストあたりのトークン上限に抵触しないよう、OCR 後のドキュメントを適切なサイズに分割して処理する仕組みを導入した。

これらの対策により、システムに投入されたドキュメントについてはスループット制限に関連するエラーが発生しないようになった。一方で、ファイルサイズが大きい場合には、分割処理やリトライの影響により処理時間が長くなる可能性がある。

レートリミット超過時のバックオフ・リトライ

スロットリングエラーを検知した場合は指数バックオフ（リトライの間隔を 1 秒→2 秒→4 秒と指数的に増やしながら再試行する方式） を適用してリトライする実装を組み込み、レートリミット超過時にも処理が継続できるよう設計した。

複数リージョン・複数モデルの分散

Bedrock のレートリミットを前提に、複数リージョンおよび複数モデル（Haiku/Sonnet 等）を併用してリクエストを分散し、同時処理可能数を増やす方針を採用した。高速モデルの活用はスループットに寄与する一方、文書特性によって精度が変動するため、品質要件とセットで最適化する。

Cloud Run によるファイル単位の並列処理

Cloud Run により、1 インスタンス＝1 ファイルとして処理を分離し、1 インスタンスでの処理件数を減らし、投入ファイル数に応じて複数インスタンスを起動し並列度を高めることでスループットを向上させる設計とした。ただし、同時起動できるインスタンス数は Cloud Run 側のクォータや最大インスタンス設定により制約されるほか、下流の OCR/LLM API のレート制限、ストレージ I/O、DB 接続数などが実効的なボトルネックとなる。 並列度を高めるほど処理時間は短縮しうる一方で、同時刻のリソース消費が増えるため実行コストが増加しうる。したがって、目標スループットとコスト、下流制約を踏まえて並列度（最大起動数）を調整する。

4.5.3　計測方法

処理性能の改善効果は、貸渡実績報告書を対象として評価した。計測指標は「総処理時間」「1 ファイルあたり平均処理時間」「スループット（ファイル/時間）」の 3 つとした。2,500 ファイルを同時投入し、処理完了までの時間を計測した。計測の詳細条件・手順については 4.9.2 節（処理性能に関する検証結果）に記載する。

4.5.4　検討した選択肢

スループット改善のアプローチとして、以下の選択肢を検討した。

表 4.18　検討した選択肢一覧

選択肢	概要	利点	課題
レートリミット引き上げ＋リトライ制御（採用）	AWS 社へのリミット引き上げ申請と、指数バックオフによる自動リトライ	既存アーキテクチャを維持したまま制約を緩和できる。実装コストが低い	AWS アカウント単位の上限であり、さらなる拡張にはアカウント分離が必要になる可能性がある

Cloud Run による並列処理（採用）	1 コンテナ＝1 ファイルの設計でスケールアウト	ファイル数に応じた柔軟なスケーリング。各ファイルが独立しているためエラー分離が容易	同時起動インスタンス数に Cloud Run 側の上限がある。インスタンス起動のオーバーヘッドが発生する
複数リージョン・複数モデル分散（採用）	Bedrock の複数リージョンと Haiku/Sonnet を併用してリクエストを分散	単一リージョンのレートリミットを回避できる	リージョン間のレイテンシ（応答遅延時間）差。モデル混在による精度変動（4.3.8 のモデル自動選定と連動）。国内リージョンの制約下で Haiku/Sonnet については Tokyo リージョンしかないため、複数リージョン利用ができない。
GPU インスタンスによる自前推論（不採用）	EC2 GPU インスタンス上で OSS モデルを自前運用	レートリミットの制約を受けない。推論パイプラインを完全にコントロールできる	GPU 環境の維持・スケーリング・障害対応の運用負荷が大きい。4.3.6 のファインチューニング検証でコスト面の課題が確認済み

4.5.5 結果

上記の対策を組み合わせた結果、2,500 ファイルの同時処理テストにおいて約 894 秒（約 15 分）で処理が完了した。これは 1 ファイルあたり平均約 0.36 秒に相当し、2024 年度の 1 ページあたり 45 秒と比較して大幅なスループット改善が確認された。（詳細は 4.9.2 節の処理性能に関する検証結果のスループット結果を参照。） 改善の主要因は以下の 2 点である。第 1 に、レートリミット引き上げ（トークン数/分を 12.5 倍、リクエスト数/秒を 5 倍に拡張）により、API スロットリングに起因するエラーおよび待機時間がほぼ解消された。第 2 に、Cloud Run によるファイル単位の並列処理により、ファイル数に応じたスケールアウトが実現し、シリアル処理に起因するボトルネックが解消された。 一方で、以下の留意点が確認された。ファイルサイズが大きい場合には、トークン上限回避のための分割処理やリトライの影響により、個々のファイルの処理時間が長くなる可能性がある。また、複数モデル分散を採用した場合、

文書特性によって精度が変動するため、4.3.8 節のモデル自動選定と連動した品質管理が必要となる。

4.5.6　結論

レートリミット引き上げ、リトライ制御、複数リージョン・複数モデル分散、および Cloud Run によるファイル単位の並列処理を組み合わせることで、2,500 ファイルを約 15 分で処理可能な水準までスループットが改善された。 2024 年度時点では数十件/日が限界であったが、本対策により数千件/時のオーダーでの処理が見込める水準に到達した。今後の課題として、Cloud Run の同時起動インスタンス数の上限に伴うスケーリング限界の評価、および大規模ファイル（数百ページ）に対する個別ファイルの処理時間最適化が残る。

4.6　課題 D：オフィスファイルの処理

本節では、課題 D（オフィスファイルの処理）に対する検証を「仮説→手法→計測方法→検討した選択肢→結果→結論」の順で整理する。

4.6.1　仮説

4.2.2 節で述べたとおり、構造化対象には PDF だけでなく Word・Excel・PowerPoint などの Office ファイルが含まれることが実証を通じて明らかになった。
実際に LibreOffice で PDF 変換を試みたところ、以下の品質劣化が確認された。

表 4.19　PDF 変換で確認された品質劣化

問題	対象	内容
フォント・図形のレイアウト崩れ	Excel / Word	Office ファイルで使用されているフォントやオートシェイプが LibreOffice と完全に互換しないため、変換後 PDF で図形の位置がずれたり消失したりする
テーブルのページ分割	Excel	テーブルがページをまたいで分割され、LLM が前後のコンテキストを把握できなくなり正確な構造化が困難になる
テキストオブジェクトの表示ズレ	Word	テキストボックス等のオブジェクトが、レンダリング差異により他のテキスト領域に重なって表示される

HTML 変換を取り入れることで、テーブルのページ分割やテキストオブジェクトの表示ズレは解消すると考えられる。しかし、HTML 変換では Excel 帳票内の丸囲み・矢印などの図形要素が消失する可能性がある。そのため、Office ファイルを構造化処理に適した中間フォーマット（PDF、HTML）に変換したうえで、図形要素については PDF 変換により既存の OCR＋LLM パイプラインとし、該当項目のみ HTML 変換の LLM パイプラインを上書きすることで、上記課題を解決することができる。

4.6.2　手法

PDF 変換

丸囲み・チェックマーク等の図形情報を画像として保持できるため、OCR＋LLM による視覚的要素の抽出に利用する。

HTML 変換

テーブル構造（要素やセル構造）を維持したまま変換できるため、表形式データやレイアウト情報の保持に利用する。横長データや複数シートにも対応可能。

最終的に、PDF 経由で取得した図形抽出結果と、HTML から生成された構造化データを統合し、1 つの構造化 JSON として出力する。

4.6.3　計測方法

Office ファイル（Excel・Word）を対象に、変換方式ごとに構造化処理を実施し、出力の精度を比較した。定量的な評価結果は 4.9.1 節の評価項目別精度検証結果一覧を参照。

4.6.4　検討した選択肢

Office ファイルの変換・処理方式として、以下の選択肢を検討した。

表 4.20　検討した選択肢一覧

選択肢	概要	利点	課題
PDF 変換のみ（不採用）	LibreOffice 等で PDF に変換し、既存の OCR＋LLM パイプラインで処理	既存パイプラインをそのまま利用可能。実装コストが低い	フォント・図形の互換性問題によるレイアウト崩れ。テーブルのページ分割。テキストオブジェクトの表示ズレ
HTML 変換のみ（不採用）	Office ファイルを直接 HTML に変換し、テーブル構造を保持したまま LLM に入力	表構造・レイアウト情報を高精度に保持。OCR を介さないため誤抽出が少ない	丸囲み・チェックマーク等の図形情報が HTML 変換では失われる
PDF 変換＋HTML 変換のハイブリッド方式（採用）	図形情報が必要な項目は PDF 経由で OCR 処理、テキスト・表構造は HTML から直接取得し、結果を統合	PDF 変換の図形保持と HTML 変換のレイアウト保持の両方を活用できる	2 つの変換パスの結果統合ロジックが必要。ファイル形式ごとの変換品質の検証が必要

4.6.5　結果

Excel・Word ファイル

ハイブリッド方式の導入により、Excel・Word ファイルに対して高精度なデータ構造化が可能であることを確認した（詳細は 4.9.1 節の評価項目別精度検証結果一覧を参照）。
具体的には以下の改善が得られた。

・テーブル構造の保持： HTML 変換により、Excel の横長テーブルや複数シート、Word の複数ページにまたがる表が、ページ分割されることなく構造化処理に投入できるようになった。PDF 変換のみの場合に発生していた表構造の認識失敗が改善された。

・図形情報の識別： PDF 変換により丸囲み等の図形情報を保持し、OCR＋LLM パイプラインで図形を識別した構造化が可能になった（詳細は 4.3.7 節を参照）。

・誤抽出の減少： 基本的なテキスト情報・テーブル情報を OCR を介さず HTML から直接取得するため、OCR 起因の誤抽出が減少し、構造化精度が向上した。

PowerPoint ファイル

PowerPoint ファイルについては、今年度の検証対象としていない。PowerPoint 固有のレイアウト（スライド単位の構成、テキストボックスの自由配置等）に対する変換・処理方式は、今後のユースケース拡大に応じて検討する。

4.6.6　結論

Office ファイル（Excel・Word）に対しては、PDF 変換のみでは品質劣化リスクが高いため、文書特性に応じて PDF 変換（図形保持）と HTML 直接変換（レイアウト保持）を併用するハイブリッド方式が有効である。 HTML 変換によりレイアウトを維持したまま構造化でき、PDF 化により丸囲み等の図形情報を OCR＋LLM パイプラインで処理できる。
本対応により、テーブルのページ分割問題、テキストオブジェクトの表示ズレ、OCR 起因の誤抽出が改善され、PDF 変換のみの方式と比較して構造化精度と汎用性が向上した。
残課題として、PowerPoint ファイルへの対応、および図形が密集する Excel 帳票での丸囲み抽出精度の改善が挙げられる。

4.7　推奨技術構成の決定

4.7.1　採用アーキテクチャ

課題 A〜D の検証結果を統合し、今年度の標準技術構成を以下のとおり決定した。

表 4.21　標準技術構成一覧

技術要素	採用構成	対応する課題・検証	採用根拠
OCR	Azure AI Vision（OCR）	課題 A（4.3.3）	全カラム平均精度 77.6%で最も安定した精度を示した

構造化 LLM	Claude Sonnet（LLM）	課題 A（4.3.3）	VLM 単体と比較して 10 ポイント以上の精度優位。処理速度・運用性も良好
打ち消し線検知	Azure Custom Vision＋Claude Sonnet（VLM）	課題 A（4.3.4）	二段判定アーキテクチャにより、打ち消し線ありページのみ VLM で再判定
プロンプト設計	Few-shot＋出力ルール明文化を全データに導入	課題 A（4.3.5）	手法追加よりも出力ルール固定が精度に最も寄与
Excel/Word 処理	PDF 変換＋HTML 変換のハイブリッド方式	課題 D（4.6）課題 A（4.3.7）	レイアウト保持と図形情報保持を両立
大規模文書	固定ページ数による自動分割（最大 20 ページ/バッチ）	課題 B（4.4）	トークン超過を防止しつつ文脈を保持
並列処理	Cloud Run（ファイル単位）＋複数リージョン分散	課題 C（4.5）	2,500 ファイルを約 15 分で処理。数千件/時のスループットを実現
モデル自動選定	特徴トリガー優先＋一致率 99%しきい値	課題 A（4.3.8）	コスト最適化と品質維持を両立するハイブリッド方式

設計思想のまとめ

今年度の標準構成は、OCR＋LLM（Azure OCR + Claude Sonnet）を基軸とし、文書特性に応じて補完的な処理を切り替える設計を採用した。VLM 単体を全面採用してパイプラインを簡素化するよりも、OCR＋LLM で広範囲を安定的に処理し、打ち消し線や丸囲みなど画像解釈が必要な箇所に限定して VLM を併用する方が、精度・コスト・運用性の総合的なバランスに優れるという検証結果に基づく判断である。

ファインチューニング（4.3.6）については、OSS VLM における有効性の示唆は得られたものの、OCR＋LLM 構成との精度差が大きく、学習データ整備や運用コストの課題もあるため、今年度は可能性調査に留めた。今後、対象範囲・費用対効果・運用性を踏まえて継続検討する。

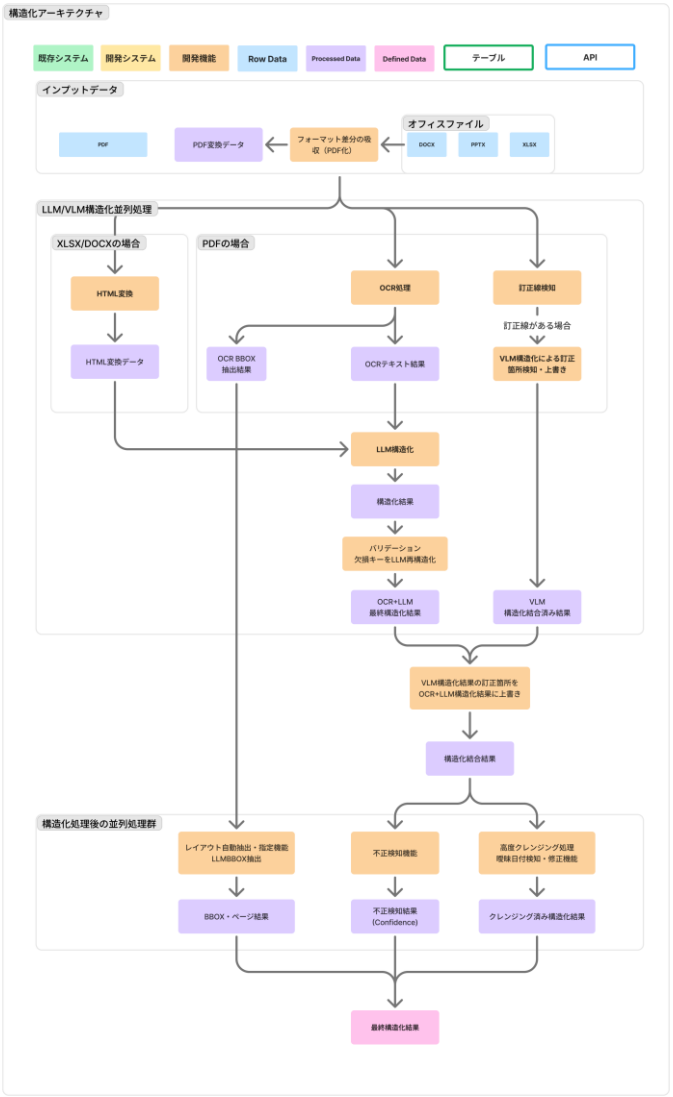


図 24 データ構造化アーキテクチャ全体図

処理の流れは以下のとおりである。まず打ち消し線（二重線を含む）の検知を行い、打ち消し線ありページを特定する。次に、全ページを対象に OCR＋LLM 処理を実行してベースとなる抽出結果を生成する。打ち消し線が検知されたページが存在する場合は、該当ページに対して VLM 処理を追加で実行する。最後に、OCR＋LLM 結果を基準として、打ち消し線で検知されたセクションに対応するフィールドのみ VLM 抽出結果で上書きし、最終的な構造化結果を確定する。この設計により、OCR＋LLM の安定性と VLM の視覚的解釈能力を適材適所で活用している。

4.8　機能・非機能要件の検証設計（KPI）

本節では、4.7 節で決定した推奨技術構成を最終的に評価するための KPI を定義した。KPI は「精度」と「処理性能」の 2 軸で構成する。

4.8.1　精度 KPI

本検証では、データ構造化における精度を多面的に評価するため、精度に関する検証項目を 24 項目（A-1～A-24）として定義した（表 31）。これらは、スキーマサジェッションの網羅性、打ち消し線検知、内容一致正解率（大量ページ・手書き・表記揺れ・単位変換・薄い文字などの難条件）、および出力の安定性といった、実運用上の品質を左右する観点を過不足なくカバーする。スキーマサジェッションはデータ構造化処理の前段階で、サンプルとして読み込んだ帳票 1 ファイルを読み込んで、どのような項目をスキーマとして設定すべきかをユーザーに提案する機能である。そのためデータ構造化機能自体の精度検証に加えて、本節の検証項目に含めた。

帳票ごとの特性差（帳票形式、ページ数、ノイズ要因等）を踏まえ、データ別にも内容一致正解率を算出する。

精度 KPI の目標値は、全 24 項目すべてを一律に「99%」としている。目標値は実運用上の品質基準として設定しており、現時点での達成状況は 4.9 節で検証する。目標値設定の背景としては、行政文書を対象としたデータ構造化結果が後続業務（集計・分析・意思決定）に直接利用されることを前提に、誤抽出や取りこぼしが業務影響や手戻りを生みやすい点を踏まえ、品質基準を高い水準で統一するためである。

なお、本評価は「完全一致」を基本とする。したがって、出力の意味が同じでも、表記ゆれや単位表現、改行や空白、括弧書き、フォーマットの差分が残る場合は減点となる。

また、本検証では LLM のモデルとして Claude 4.5 を採用し検証を実施した。背景には、それまで使用していた Claude 3.5 に代わり、本年度の実証期間中に最新モデルである Claude 4.5 が公開されたため、これを機に Claude 4.5 へ切り替えたという経緯がある。

表 4.22　設定した KPI 一覧

No.	評価項目種類	評価項目	説明	検証粒度	対象データ
A-1	スキーマサジェッション	スキーマサジェッション含有率	EBPM 用に作成したスキーマに対して、スキーマサジェッションで抽出できたスキーマの割合（意味が同じものも含む）	資料単位	各実証単位からそれぞれ任意の 1 データ

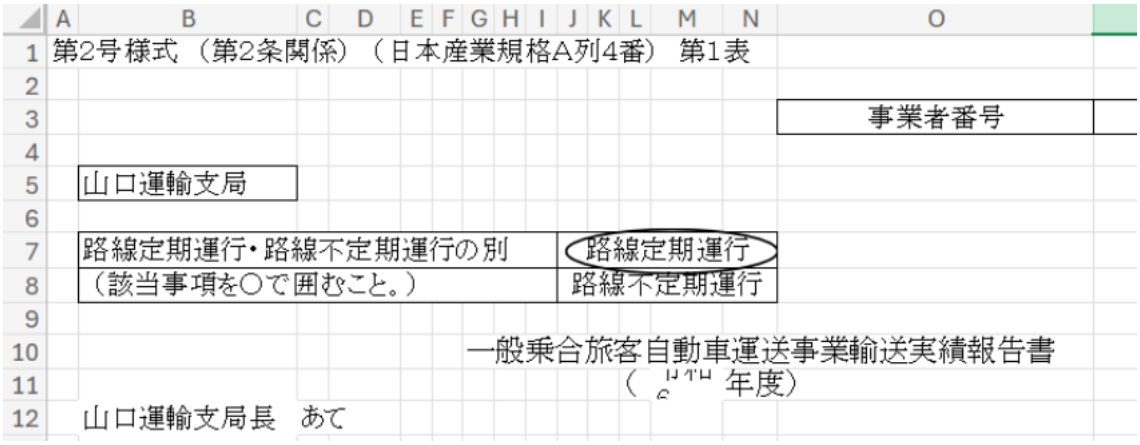
A-2	スキーマサジェッション	大量ページデータに対するスキーマサジェッション含有率	10 ページ以上の大量データを対象とした場合のスキーマサジェッション含有率	ファイル単位	貨物自動車運動事業報告規則に基づく事業報告書(1 号様式、2 号様式、3 号様式、貸借対照表、損益計算書)
A-3	前処理	打ち消し線検知率	打ち消し線の有無を正しく検知できた割合（再現率）	セル単位	貨物自動車運送事業報告規則に基づく事業実績報告書（4 号様式 貨物自動車運送事業 事業実績報告書）、自動車事故報告書別記様式
A-4	データ構造化課題	大量ページ正解率	10 ページ以上あるファイルのみを対象にした場合の正解率	ファイル単位	貨物自動車運動事業報告規則に基づく事業報告書(1 号様式、2 号様式、3 号様式、貸借対照表、損益計算書)
A-5	データ構造化課題	Excel（丸囲み）正解率	丸囲みがあるカラムのみの正解率	カラム単位	一般乗合旅客自動車運送事業輸送実績報告書
A-6	データ構造化課題	手書き	手書きがあるセルのみを対象にした場合の正解率	セル単位	貨物自動車運送事業報告規則に基づく事業実績報告書（4 号様式 貨物自動車運送事業 事業実績報告書）、自家用自動車有償貸渡事業 貸渡実績報告書（レンタカー事業）【様式 2】事務所別車種別配車両数一覧
A-7	データ構造化課題	フォーマット違い	同じ情報でありながら別フォーマットで提出されているファイルのみを対象にした場合の正解率（例：共通様式でないもの、事業者により異なる貸借対照表等）	資料単位	貨物自動車運動事業報告規則に基づく事業報告書(1 号様式、2 号様式、3 号様式、貸借対照表、損益計算書）、鉄道事業報告書

A-8	データ構造化課題	複数様式混在	1～4 号様式がまとめて提出されているなど、複数様式が 1PDF になっているファイルのみを対象にした場合の正解率	資料単位	貨物自動車運動事業報告規則に基づく事業報告書(1 号様式、2 号様式、3 号様式、貸借対照表、損益計算書）
A-9	データ構造化課題	図形認識	丸囲みやチェックボックスなどがある場合のカラムのみを対象に正解率を算出	カラム単位	貨物自動車運動事業報告規則に基づく事業報告書(1 号様式、2 号様式、3 号様式、貸借対照表、損益計算書）
A-10	データ構造化課題	表構造（フラット）	表がある場合、表のカラムのみを対象に正解率を算出	カラム単位	貨物自動車運動事業報告規則に基づく事業報告書(1 号様式、2 号様式、3 号様式、貸借対照表、損益計算書）自家用自動車有償貸渡事業 貸渡実績報告書（レンタカー事業）【様式 2】事務所別車種別配車両数一覧
A-11	データ構造化課題	資料ごとの表記揺れへの対応	同一の情報でありながら表記揺れが生じている際に同一スキーマ・プロンプトで抜き出せたセルのみを対象にした場合の正解率（例：現金、現預金などの項目名の表記揺れ）	カラム単位	貨物自動車運動事業報告規則に基づく事業報告書(1 号様式、2 号様式、3 号様式、貸借対照表、損益計算書）、鉄道事業報告書
A-12	データ構造化課題	和暦表記の日付情報を西暦に変換	和暦表記の日付情報があるセルのみを対象に正解率を算出	カラム単位	貨物自動車運送事業報告規則に基づく事業実績報告書（4 号様式 貨物自動車運送事業事業実績報告書）、国会答弁書データ

A-13	データ構造化課題	A4 サイズに収まらない Excel 資料	横方向に 100 列以上記載されているなど、縦横比が A4 サイズに収まっていないファイルのみを対象にした場合の正解率	ファイル単位	倉庫業法第 27 条に基づく倉庫事業経営状況報告（倉庫事業経営指標 調査票）
A-14	データ構造化課題	セル内で記載されている文字列が見切れている Excel 資料	セル内改行や長文が原因で文字列が見切れているセルのみを対象にした場合の正解率	セル単位	「内航船省エネルギー格付制度」事務取扱要領様式第 1、第 2、第 3、第 4
A-15	データ構造化課題	罫線が入っていない表形式	罫線なし表形式のデータのみを対象にした正解率	カラム単位	貨物自動車運動事業報告規則に基づく事業報告書(1 号様式、2 号様式、3 号様式、貸借対照表、損益計算書）
A-16	データ構造化課題	項目名が記載されていない表	財務情報など科目の金額・小計・合計などの項目名（出力スキーマ名称に相当）が記載されていない場合のセルのみを対象にした正解率	カラム単位	貨物自動車運動事業報告規則に基づく事業報告書(1 号様式、2 号様式、3 号様式、貸借対照表、損益計算書）
A-17	データ構造化課題	単位の統一	円と千円など資料によって単位が異なるセルのみを対象にした場合の正解率（単位統一した状態で数値を抜き出せるか）	カラム単位	観光地域づくり法人形成・確立計画、貨物自動車運動事業報告規則に基づく事業報告書(1 号様式、2 号様式、3 号様式、貸借対照表、損益計算書）
A-18	データ構造化課題	文字が薄い	PDF 形式のファイルにおいて文字が薄いファイルのみを対象にした場合の正解率	セル単位	貨物自動車運送事業報告規則に基づく事業実績報告書（4 号様式 貨物自動車運送事業 事業実績報告書）

A-19	ファイル形式	Excel（単一シート）	ファイル種類が Excel 形式（単一シート）の場合の正解率	資料単位	該当形式の全データ
A-20	ファイル形式	Excel（複数シート）	ファイル種類が Excel 形式（複数シート）の場合の正解率	資料単位	該当形式の全データ
A-21	ファイル形式	Word	ファイル種類が Word 形式の場合の正解率	資料単位	該当形式の全データ
A-22	ファイル形式	PPT	ファイル種類が PPTX 形式の場合の正解率	資料単位	該当形式の全データ
A-23	ファイル形式	PDF	ファイル種類が PDF 形式の場合の正解率	資料単位	該当形式の全データ
A-24	運用課題	出力の安定性	同一データ・同一プロンプトで結果の揺れが生じないか。10 回実行して同じ結果を得られるかを確認	ファイル単位	自家用自動車有償貸渡事業 貸渡実績報告書（レンタカー事業）【様式 2】事務所別車種別配車両数一覧　貨物自動車運送事業報告規則に基づく事業実績報告書（4 号様式：貨物自動車運送事業事業実績報告書）

設定した各 KPI のうち、評価項目種類がデータ構造化課題かつ検証粒度が資料単位でないものに対する実際の帳票例を示す。



第2号様式（第2条関係）（日本産業規格A列4番） 第1表

事業者番号	

山口運輸支局

路線定期運行・路線不定期運行の別	路線定期運行
（該当事項を○で囲むこと。）	路線不定期運行

一般乗合旅客自動車運送事業輸送実績報告書

（ 年度）

山口運輸支局長 あて

図 25 A-5 Excel（丸囲み）正解率の帳票例

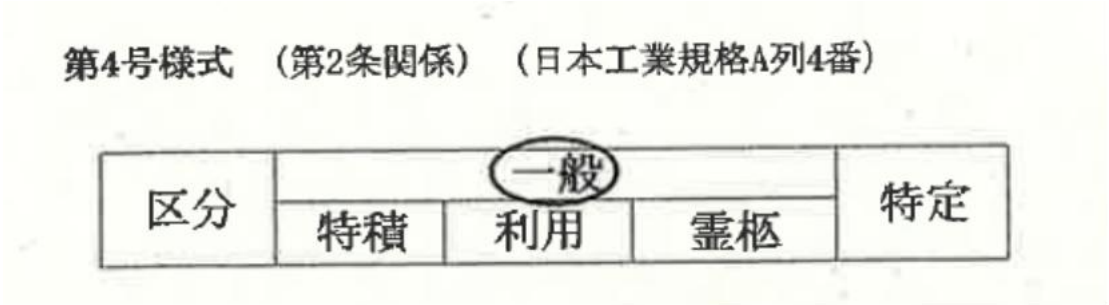
輸送実績（前年4月1日から3月31日まで）

	延実在車両数（日車）	延実働車両数（日車）	走行キロ（キロメートル）	実車キロ（キロメートル）	輸送トン数 実運送（トン）	輸送トン数 利用運送（トン）	営業収入（千円）
北海道							
東北							
北陸信越							
関東							
中部							
近畿							
中国							
四国							
九州							
沖縄							
全国計							

事故件数（前年4月1日から3月31日まで）

交通事故件数	0	重大事故件数	0	死者数	0	負傷者数	0

図 26 A-6 手書きの帳票例



第4号様式 （第2条関係） （日本工業規格A列4番）

区分	一般			特定
	特積	利用	霊柩	

図 27 A-9 図形認識の帳票例

輸送実績（前年4月1日から3月31日まで）

	延実在車両数（日車）	延実働車両数（日車）	走行キロ（キロメートル）	実車キロ（キロメートル）	輸送トン数		営業収入（千円）
					実運送（トン）	利用運送（トン）	
北海道							
東北							
北陸信越							
関東							
中部							
近畿							
中国							
四国							
九州	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
沖縄							
全国計	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

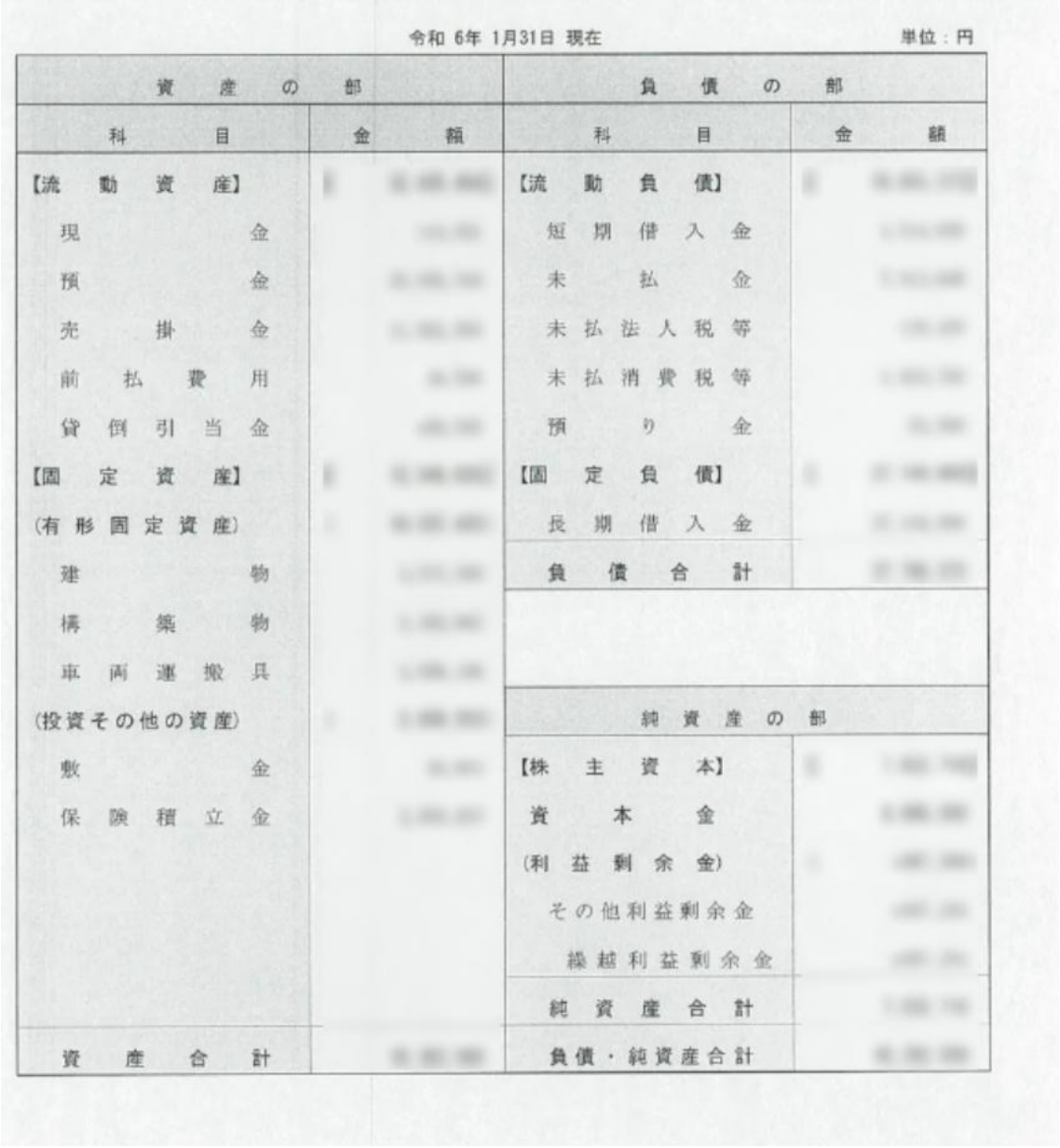
事故件数（前年4月1日から3月31日まで）

交通事故件数	0	重大事故件数	0	死者数	0	負傷者数	0

図 28 A-10 表構造（フラット）帳票例

（単位：千円）

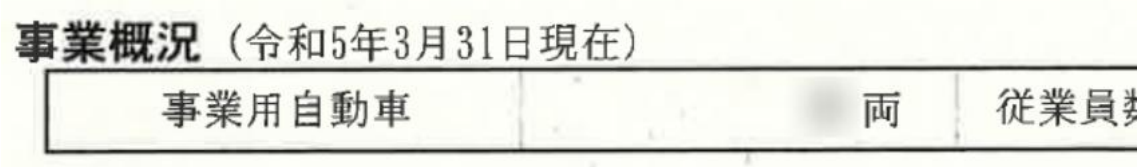
科目	金額	科目	金額
（資産の部）		（負債の部）	
Ⅰ．流動資産		Ⅰ．流動負債	
現金預金		支払手形	
受取手形		買掛金	
未収運賃		短期借入金	
有価証券		1年以内返済予定の長期借入金	
商品		1年以内償還予定社債	
貯蔵品		未払金	
前払費用		未払費用	
前払金		未払法人税等	
未収還付消費税等		未払消費税等	
未収収益		前受金	
短期貸付金		預り金	
立替金			
仮払金		賞与引当金	
		繰延税金負債	
繰延税金資産		その他流動負債	
その他流動資産		《流動負債合計》	
貸倒引当金		Ⅱ．固定負債	
《流動資産合計》		社債	
Ⅱ．固定資産		長期借入金	
1．有形固定資産		退職給付引当金	
車両運搬具		役員退職慰労引当金	
建物		預り保証金	
構築物		繰延税金負債	
機械装置		その他固定負債	
工具器具備品		《固定負債合計》	
リース資産		負債の部合計	
土地		（純資産の部）	
建設仮勘定		Ⅰ．株主資本	
（有形固定資産合計）		資本金	
2．無形固定資産		新株式申込証拠金	
のれん		資本剰余金	
ソフトウェア		資本準備金	
電話加入権		その他資本剰余金	
（無形固定資産合計）		（資本剰余金合計）	
3．投資その他の資産		利益剰余金	
投資有価証券		利益準備金	
関係会社株式		任意積立金	
出資金		その他利益剰余金	
長期貸付金		（利益剰余金合計）	
長期前払費用		自己株式	
差入保証金		自己株式申込証拠金	
保険積立金		《株主資本合計》	
預託金		Ⅱ．評価・換算差額等	
貸倒引当金		その他有価証券評価差額金	
（投資その他の資産合計）		土地再評価差額金	
《固定資産合計》		繰延ヘッジ損益	
Ⅲ．繰延資産		《評価・換算差額合計》	
		Ⅲ．新株予約権	
《繰延資産合計》		純資産の部合計	
資産の部合計		負債の部・純資産の部合計	



令和 6年 1月31日 現在　　単位：円

資産の部		負債の部	
科目	金額	科目	金額
【流動資産】		【流動負債】	
現金		短期借入金	
預金		未払金	
売掛金		未払法人税等	
前払費用		未払消費税等	
貸倒引当金		預り金	
【固定資産】		【固定負債】	
(有形固定資産)		長期借入金	
建物		負債合計	
構築物			
車両運搬具			
(投資その他の資産)		純資産の部	
敷金		【株主資本】	
保険積立金		資本金	
		(利益剰余金)	
		その他利益剰余金	
		繰越利益剰余金	
		純資産合計	
資産合計		負債・純資産合計	

図 29・30 A-11 資料ごとの表記揺れ帳票例



事業概況（令和5年3月31日現在）

事業用自動車	両	従業員数

図 31 A-12 和暦表記の日付情報の帳票例

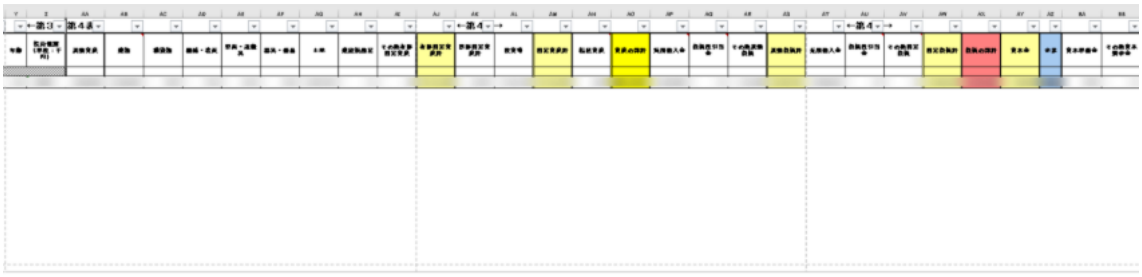


図 32 A-13 A4 サイズに収まらない Excel 資料の帳票例

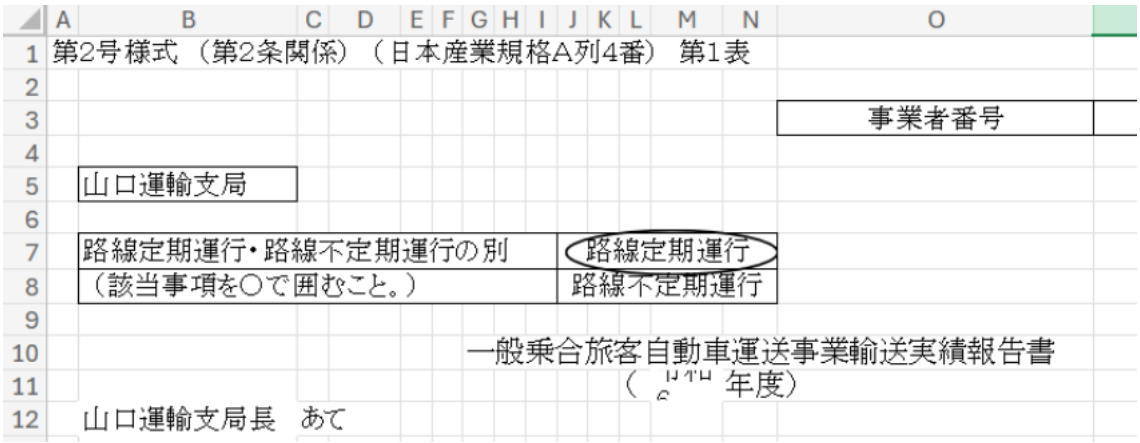
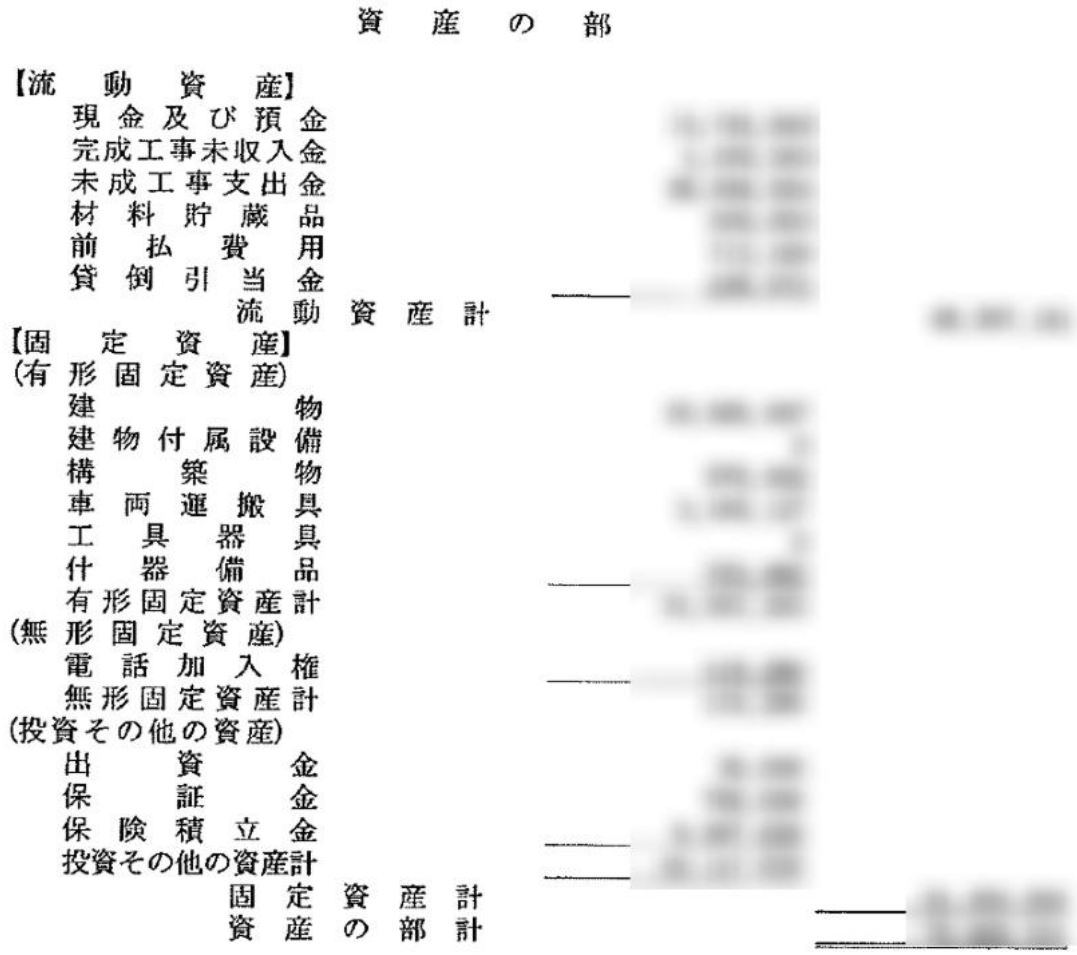


図 33 A-14 セル内で記載されている文字列が見切れている Excel 資料の帳票例


資　産　の　部

【流　動　資　産】
現金及び預金
完成工事未収入金
未成工事支出金
材料貯蔵品
前払費用
貸倒引当金
流動資産計
【固　定　資　産】
(有形固定資産)
建物
建物付属設備
構築物
車両運搬具
工具器具
什器備品
有形固定資産計
(無形固定資産)
電話加入権
無形固定資産計
(投資その他の資産)
出資金
保証金
保険積立金
投資その他の資産計
固定資産計
資産の部計

図 34 A-15 罫線が入っていない表形式の帳票例

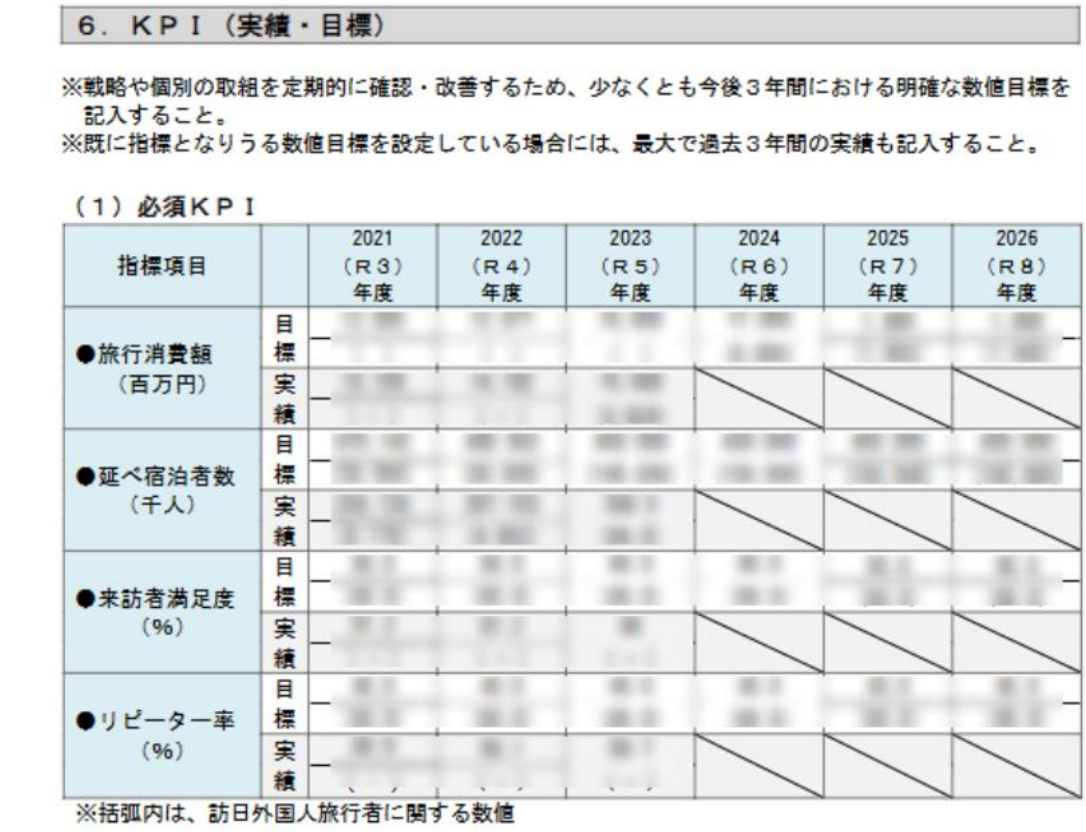
資　産　の　部

【流　動　資　産】
現金及び預金
完成工事未収入金
未成工事支出金
材料貯蔵品
前払費用
貸倒引当金
流動資産計
【固　定　資　産】
(有形固定資産)
建物
建物付属設備
構築物
車両運搬具
工具器具
什器備品
有形固定資産計
(無形固定資産)
電話加入権
無形固定資産計
(投資その他の資産)
出資金
保証金
保険積立金
投資その他の資産計
固定資産計
資産の部計

図 35 A-16 項目名が記載されていない表の帳票例

指標項目		2020（R２）年度	2021（R３）年度	2022（R４）年度	2023（R５）年度	2024（R６）年度	2025（R７）年度
●旅行消費額（一人あたり／円）	目標	( )	( )	( )	( )	( )	( )
	実績	( )	( )	( )			
●延べ宿泊者数（千人）	目標	(0.2)	(0.2)	(0.6)	(1.2)	(1.5)	(1.8)
	実績	(0.5)	(0.4)	(0.8)			
●来訪者満足度（%）	目標	( )	( )	( )	( )	( )	( )
	実績	( )	( )	( )			
●リピーター率（%）	目標	( )	( )	( )	( )	( )	( )
	実績	( )	( )	( )			

※括弧内は、訪日外国人旅行者に関する数値



6．ＫＰＩ（実績・目標）

※戦略や個別の取組を定期的に確認・改善するため、少なくとも今後３年間における明確な数値目標を記入すること。
※既に指標となりうる数値目標を設定している場合には、最大で過去３年間の実績も記入すること。

（１）必須ＫＰＩ

指標項目		2021（R３）年度	2022（R４）年度	2023（R５）年度	2024（R６）年度	2025（R７）年度	2026（R８）年度
●旅行消費額（百万円）	目標						
	実績						
●延べ宿泊者数（千人）	目標						
	実績						
●来訪者満足度（%）	目標						
	実績						
●リピーター率（%）	目標						
	実績						

※括弧内は、訪日外国人旅行者に関する数値

図 36・37 A-17 単位の統一がなされていない帳票例

輸送実績（前年4月1日から3月31日まで）

	延実在車両数（日 車）	延実働車両数（日 車）	走行キロ（キロメートル）	実車キロ（キロメートル）	輸送トン数 実運送(トン)	輸送トン数 利用運送(トン)	営業収入（千円）
北海道							
東北							
北陸信越							
関東							
中部							
近畿							
中国							
四国							
九州							
沖縄							
全国計							

図 38 A-18 文字が薄い帳票例

4.8.2 処理性能 KPI

処理性能 KPI は、「現実的な時間内で処理が完了し、かつ同時並列処理により必要な処理量をさばけること」を確認するための指標である。処理性能 KPI は、インフラ構成や同時実行数・リトライ・分割戦略などの運用条件により変動しうるため、測定時はテスト条件を明確にしたうえで再現可能な手順で取得する。

表 4.23　処理性能 KPI 一覧

No.	評価指標	目標値	計測方法
1	1 件あたり処理時間	44.5 秒未満/頁	実行ログのタイムスタンプ差分
2	スループット（同時処理件数）	500 ファイル/時	負荷テスト結果

4.8.3　KPI 算出方法の詳細

(1)スキーマサジェッション含有率（A-1・A-2 対象）

定義： システムが自動生成したスキーマ（サジェッション）が、EBPM 用データ構造化スキーマとして定義したカラムにどの程度含まれているかの割合。

$$スキーマサジェッション含有率 = \left(サジェッションに含まれていた定義カラム数/定義カラム総数\right) \times 100$$

判定基準： サジェッションされたカラム名が定義カラムと一致または意味的に同等の場合「含有」、対応するサジェッションが存在しない場合「非含有」とした。

計算例：

定義カラム: 10 カラム（事業者名,住所,車両台数,営業開始日, ...）　サジェッション結果: 8 カラムがマッチング（住所→所在地、車両台数→保有台数 等は意味的に同等と判断）

$$スキーマサジェッション含有率 = (8/10) \times 100$$

(2)検知正解率（A-3 対象）

定義： 打ち消し線が含まれると判定されたデータのうち、実際に正しく判定できたものの割合（再現率）。

$$検知正解率 = TP/(TP + FN) \times 100$$

表 4.24　打ち消し線検知指標表

分類	元資料の状態	システム出力	判定
TP	打ち消し線あり	打ち消し線ありと検知	正解
FN	打ち消し線あり	打ち消し線なしと判定（見逃し）	不正解

評価単位： 1 レコード × 1 フィールド= 1 評価単位

計算例：

評価対象: 8 レコード × 6 フィールド（打ち消し線評価対象）= 48 評価単位、結果: TP=40, FN=8

$$検知正解率 = 40/(40 + 8) \times 100$$

(3)内容一致正解率（A-4～A-23 対象）

定義： 構造化結果と元資料を照合し、抽出内容が正しい割合。

$$検知正解率 = TP/(TP + FN) \times 100$$

判定基準（フィールド単位）：

表 4.25　内容一致正解率算出表

分類	元資料の状態	システム出力	判定
TP	記載あり	正しい値を抽出	正解
FP	記載あり	誤った値を抽出、 または未抽出（空）	不正解
TN	記載なし （空欄・該当なし）	未抽出（空）	正解
FN	記載なし （空欄・該当なし）	何らかの値を抽出 （ハルシネーション）	不正解

評価単位： 1 レコード × 1 フィールド= 1 評価単位

計算例：

評価対象: 50 レコード × 10 フィールド= 500 評価単位、結果: TP=380, FP=50, TN=60, FN=10

内容一致正解率=(380+60) / (380+60+50+10)×100

(4)出力の安定性（A-24 対象）

定義： データ構造化結果が、同一データ・同一プロンプトで実施した場合に結果の揺れが生じないかを確認。10 回実行して同じ結果を得られるかを確認。

出力の安定性=10 回実施して同一の内容であったカラム数定義したカラム総数×100

計算例：

定義したスキーマ: 10 カラム、10 回実施して 9 カラムが同一内容

出力の安定性=910×100

(5) 1 件あたり処理時間（処理性能 KPI No.1）

定義： 1 ページのデータを構造化するのに要する平均時間。

1 件あたり処理時間=総処理時間（秒）処理ページ数

表 4.26　処理工程一覧

処理工程	目標時間	計測方法
OCR 処理	9.5 秒/頁	OCR 開始～完了のタイムスタンプ差分
構造化処理・ 自動評価処理	35 秒/頁	LLM 呼び出し開始～完了・評価処理開始～ 完了のタイムスタンプ差分
合計	**44.5 秒/頁**	

計算例： 処理対象 100 ページ、総処理時間 4,200 秒の場合：

1 件あたり処理時間=4,200/100=42 秒/頁

(6)スループット（処理性能 KPI No.2）

定義： 単位時間あたりに処理できるデータ件数、または同時に処理可能な最大件数。

スループット=処理完了件数/処理時間（時間）

計算例： 1,000 件を同時投入、2 時間で全件完了の場合：

スループット=1,000/2=500 件/時間

4.9　KPI 検証結果

4.9.1　精度に関する検証結果

検証の結果、多くのデータでは内容一致正解率は概ね高水準であった。特にセル数が大きいデータでも 99%台後半の精度が確認できている。表構造が比較的明確で、スキーマと入力データの対応関係が安定しているケースでは、OCR＋LLM による構造化は大規模データに対しても安定して動作する傾向が見られた。

一方で、データによっては 95%前後、あるいは 90%台前半から 80%台の結果も見られ、精度にはばらつきが存在した。

表 4.27　データ別精度検証結果一覧（ claude 4.5 sonnet を使用）

データ番号	名前	全件数（セル単位）	内容一致正答数（セル単位）	内容一致正答率
1103	観光地域づくり法人形成・確立計画	34,240	33,568	98.04%
1201	貨物自動車運送事業報告規則に基づく事業実績報告書 （4 号様式：貨物自動車運送事業事業実績報告書）	1,020	978	95.88%
1203	貨物自動車運動事業報告規則に基づく事業報告書 (1 号様式、2 号様式、3 号様式、貸借対照表、損益計算書）	18,040	17,185	95.26%
1703	内航海運事業に係る報告書 （内航海運業事業概要報告書）【紙】	1,680	1,645	97.92%
1707	「内航船省エネルギー格付制度」 事務取扱要領様式第 1、第 2、第 3、第 4	25,718	24,304	94.50%

1708	船員法第 111 条に基づく事業状況に関する報告	248,670	247,427	99.50%
1901	倉庫業登録申請書（倉庫業登録申請書）	20,541	20,293	98.79%
1906	期末倉庫使用状況報告書（8 号様式）	156,996	156,961	99.98%
1908	受寄物入出庫高および保管残高報告書（9 号様式）	92,871	92,710	99.83%
1911	【様式第 1 号】総合効率化計画認定申請書	12,860	12,823	99.71%
1910	倉庫業法第 27 条に基づく倉庫事業経営状況報告（倉庫事業経営指標 調査票）	3,560	3,446	96.80%
1912	【様式第 6 号】流通業務総合効率化事業実施状況報告書	1,344	1,263	93.97%
1913	【第 1 号様式】新・増設倉庫証明申請書	966	811	83.95%
2001	自動車事故報告書別記様式（第 3 条関係）	4,326	4,103	94.85%
2201	国会答弁書データ	300	272	90.67%
2403	補助事業に関する情報： 応募様式・交付申請書・交付決定通知書・完了実績報告書・額の確定通知書（・補助金台帳）	778	753	96.79%
2401	要望書	120	102	85.00%
2501	自家用自動車有償貸渡事業 貸渡実績報告書（レンタカー事業）【様式 1】貸渡実績報告書	8,300	8,134	98.00%
2502	自家用自動車有償貸渡事業 貸渡実績報告書（レンタカー事業）【様式 2】事務所別車種別配車両数一覧	9,700	9,573	98.69%
2503	レンタカー協会 PDF	15,810	15,810	100.00%

3305	事業概況報告書（鉄道事業設備投資、その他の事業設備投資、貸借対照表、損益計算書）、 事業実績報告書（営業収益表、役職員数および職員給与額表、旅客輸送実績表、貨物輸送実績表、走行キロ表、電力および燃料表、トンネルおよび橋りょう表、踏切道および立体交差表、車両数表）	1,164,188	1,163,982	99.98%
3303	事業報告書	211,276	210,983	99.86%
3301	運転事故等届出書（第 2 号様式）または 運転事故等報告書（第 1 号様式）	3,660	3,465	94.67%
3302	災害報告書（第 6 号様式）	364	357	98.08%
3402	一般乗用旅客自動車運送事業 輸送実績報告書	7,880	7,813	99.15%
3408	一般貸切旅客自動車運送事業 輸送実績報告書	159,092	158,930	99.90%
3407	一般乗合旅客自動車運送事業 輸送実績報告書	159,092	158,930	99.90%
3419	自家用有償旅客運送輸送実績報告書	2,088	1,926	92.24%
3413	自家用有償観光旅客等運送登録申請書等	115,884	115,842	99.96%
3401	一般乗用旅客自動車運送事業 事業報告書	17,740	17,495	98.62%
3406	一般乗合旅客自動車運送事業 事業概況報告書	256,600	255,082	99.41%
3410	一般貸切旅客自動車運送事業 事業概況報告書	256,600	255,082	99.41%
3501	自治体、バス事業者、鉄道事業者に、 GTFS、新モビリティ等の導入状況について、運輸局経由で実施しているアンケート	1,489	1,483	99.60%
3502	自治体等の計画書を全国から収集： 地域公共交通計画	1,884	1,590	84.39%
3701	UR 事故報告書データ	1,302	1,153	88.56%

3801	建設工事施工統計調査_法人版	23,081	22,720	98.44%
3802	建設工事施工統計調査_個人版	23,081	22,720	98.44%

次に設定した KPI 一覧（A-1～A-24）別のデータ構造化精度（正解率）は以下の通りである。

表 4.28　評価項目別精度検証結果一覧

No.	評価項目	正解率	考察・示唆	達成状況
A-1	スキーマ サジェッション 含有率	70.77%	サジェッションのみで EBPM 用スキーマを網羅することは難しく、スキーマテンプレート整備や過去スキーマの再利用との組み合わせが必要	未達成
A-2	大量ページデータに対するスキーマ サジェッション 含有率	69.81%	ページ数増加に伴い情報分散・表現揺れが増え、サジェッションの取りこぼしが生じやすい。	未達成
A-3	打ち消し線検知率	37%～ 34%	打ち消し線の視認性がスキャン品質により変動するため、二段判定と学習データ拡充が必須	未達成
A-4	大量ページ正解率	95.86%	多ページ文書でも一定精度で構造化可能。99%到達には分割戦略・ページ単位再実行の改善余地がある	未達成
A-5	Excel（丸囲み） 正解率	94.4%	現行方式で一定の精度を確認	未達成
A-6	手書き	78.46% ～ 93.83%	手書き品質・筆記具・解像度により結果が大きく変動。手書きに強い OCR/VLM の部分適用が有効	未達成
A-7	フォーマット違い	95.26% ～ 99.86%	比較的高い耐性を確認。95%台のケースは帳票間で項目位置や表現が変わる影響が残る	一部達成
A-8	複数様式混在	95.26% ～ 98.69%	フォーマット判定（分類）→最適プロンプト/抽出ルール選択の強化が有効	未達成
A-9	図形認識	100%	対象範囲の図形については現行手法で高精度	達成

A-10	表構造 （フラット）	92.43% ～ 98.69%	列見出しの階層構造・結合セル・罫線有無などの条件差に対応した補正が必要	未達成
A-11	資料ごとの 表記揺れへの対応	93.58% ～ 100%	表記揺れへの一定の耐性は確認。辞書（同義語・表記揺れ）やスキーマ側のエイリアス設計の強化が有効	一部達成
A-12	和暦→西暦変換	100%	日付正規化処理は有効に機能	達成
A-13	A4 サイズに収まらない Excel 資料	96.80%	横長資料は縮小・分割に起因する文字潰れや列対応ずれが起きやすい。レンダリング設定の改善が必要	未達成
A-14	セル内で記載されている文字列が見切れている Excel 資料	100%	HTML を用いた構造化によって精度改善	達成
A-15	罫線なし表	75.78%	表領域・列境界の推定が困難。罫線補完・セル境界推定などの前処理強化が必要	一部達成
A-16	項目名なし表	87.78%	周辺テキスト（注記・見出し）を含めた抽出戦略が必要	未達成
A-17	単位の統一	93.34% ～ 98.29%	単位表記の位置や換算ルールの曖昧さにより誤りが残りうる	未達成
A-18	文字が薄い	65%	OCR 段階での欠落・誤認識が支配的。画像補正（コントラスト強調等）の対策が必要	未達成
A-19	Excel	99.61%	テキスト量が多くなると表記揺れに対する抽出精度が低下する傾向にあるため、分割処理などを検討	達成
A-20	Excel （複数シート）	99.61%	昨年度対応できなかった複数シート Excel に対しても高精度の情報抽出が可能であることを確認した。テキスト量が多い場合には分割処理などを検討	達成
A-21	Word	99.94%	HTML を用いたデータ構造化により精度向上。画像情報に対しては未抽出が見られた。	達成

A-22	PPT	97.10%	テキスト情報の抽出精度は高かったものの図表の読み取りに課題あり	未達成
A-23	PDF	99.62%	丸囲みや 10 ページ以上の大量データに対するデータ構造化にて、誤抽出が見られた。章構成の理解を行う処理を入れるなど、段階的に情報抽出を行うよう検討	達成
A-24	出力の安定性	99.67%~99.97%	セル単位で安定した出力結果となっていることが確認できた。逆説的に、誤った結果も繰り返し出力される可能性があるので、データ構造化精度向上が求められる。	達成

結果を見ると、高い精度が出やすい項目と、精度が落ちやすい項目に傾向があった。高い精度が出やすいのは、文書内の位置やラベルが比較的固定されていて、使われる語彙や形式が限定される項目である。このタイプは OCR＋LLM 構成でも安定しやすかった。

一方で、精度が低下しやすい項目にはいくつか共通点があった。例えば、章立ての構成も資料によって異なる帳票は抽出範囲がずれやすいほか帳票間の記載ゆれや略語、レイアウト差が大きい項目も、同じプロンプトで拾いにくい。表や複雑な階層構造となっている資料など、レイアウト依存で対応関係の解釈が必要な項目では、値とキーの対応を取り違えやすい。

さらに、丸囲み、手書き、薄い文字、かすれなど視覚ノイズが強い場合は、認識のブレがそのまま精度低下につながる。打ち消し線や追記がある項目では、旧情報と新情報の切り分けが必要になり、誤りが増えやすかった。

低精度の評価項目の誤りは、いくつかのパターンに整理できる。まず、数字や漢字の誤認識、薄い印字やかすれによる OCR 誤読が混入する。次に、自由記述欄で抽出範囲の開始・終了がずれる、改行結合や箇条書き境界の扱いが崩れる、といった境界の誤りが起きやすい。表や注記、脚注、矢印や括弧による参照関係では、対応付けの誤りが出やすい。選択式項目では、丸囲みやチェックの見落とし、非選択肢の混入、複数選択の数の不一致が典型的である。訂正・追記を含む項目では、打ち消し線で消された旧情報が残る、追記情報を取りこぼす、旧情報と新情報を連結してしまうといった誤りが見られた。

以上を踏まえると、改善は評価項目のタイプごとに分けて対応するのが考えられる。文章抜き出し系は、プロンプトだけでは改善が頭打ちになりやすいので、OCR 前処理（画像補正）、抽出対象領域の切り出し、改行・空白の正規化を優先する。丸囲みや打ち消し線を含む項目は、OCR＋LLM の弱点になりやすいため、打ち消し線検知による二段判定や、視覚入力（VLM）による補正の強化をする。表やレイアウト依存の項目は、表再構成とキー対応の誤りが支配的になりやすいので、表検出結果の活用、セル単位の整形ルール、失敗時の再試行（別モデル・別プロンプト）などを組み込む余地がある。

4.9.2　処理性能に関する検証結果

以下のデータ別処理性能検証結果一覧から、データごとの処理特性に明確な傾向が見られた。スキーマサジェッション処理時間は、最短で 14 秒(2503:レンタカー協会 PDF)から最長で 106 秒(1203:貨物自動車運動事業報告規則に基づく事業報告書(1 号様式、2 号様式、3 号様式、貸借対照表、損益計算書）)まで幅があり、データ構造化の 1 件あたり処理時間は最短で 50.6 秒(2502:自家用自動車有償貸渡事業 貸渡実績報告書（レンタカー事業）【様式 2】事務所別車種別配車両数一覧)から最長で 1,436.8 秒(1103:観光地域づくり法人形成・確立計画)となった。

処理時間の長短を決定する主要因として、1 ファイルあたりの平均ページ数が大きく影響していることが確認できた。1103:観光地域づくり法人形成・確立計画は平均 23 ページで 1,436.8 秒、1901:倉庫業登録申請書（倉庫業登録申請書）は平均 33.14 ページで 224.7 秒、1911:【様式第 1 号】総合効率化計画認定申請書は平均 29.95 ページで 1,023.0 秒と、ページ数が多いほど処理時間が増加する傾向にある。一方、1 ページのみの帳票である 1201:貨物自動車運送事業報告規則に基づく事業実績報告書（4 号様式：貨物自動車運送事業事業実績報告書）(130.4 秒)や自家用自動車有償貸渡事業 貸渡実績報告書（レンタカー事業）の各様式(50.6〜134.5 秒)では比較的短時間で処理が完了している。

ただし、ページ数と処理時間の関係は単純な比例関係ではない。1901:倉庫業登録申請書（倉庫業登録申請書）は平均 33.14 ページで 224.7 秒(約 6.8 秒/ページ)であるのに対し、1911:【様式第 1 号】総合効率化計画認定申請書は平均 29.95 ページで 1,023.0 秒(約 34.2 秒/ページ)と、ページあたりの処理時間に約 5 倍の差が生じている。これは、出力スキーマの項目の数や複雑さ、帳票の複雑さ、記載項目数、レイアウトの多様性、自由記述欄の有無などが処理負荷に影響を与えていることを示唆している。

スキーマサジェッション処理時間についても同様の傾向が見られる。最も長い 1203:貨物自動車運動事業報告規則に基づく事業報告書(1 号様式、2 号様式、3 号様式、貸借対照表、損益計算書）(106 秒)は 11.1 ページの複数様式を含む複合的な帳票であり、帳票種別の多様性がスキーマ推定の負荷を高めていると考えられる。一方、単一様式で構造が比較的単純な自家用自動車有償貸渡事業 貸渡実績報告書（レンタカー事業）の各帳票では 14〜25 秒と短時間で処理が完了している。

全体として、目標時間 44.5 秒/ページを大きく下回る処理性能が達成されており、特にページ数の少ない帳票では 10〜20 秒/ページ程度の高速処理が実現できている。ただし、ページ数が多く構造が複雑な帳票では、1 ページあたり 30 秒以上を要する場合もあり、今後の改善余地がある。

表 4.29　データ別処理性能検証結果一覧

データ番号	名前	スキーマサジェッション処理時間(単位：秒)	データ構造化１件あたり処理時間(単位：秒)	1 ファイルの平均ページ数
1103	観光地域づくり法人形成・確立計画	39	1,436.8	23
1201	貨物自動車運送事業報告規則に基づく事業実績報告書（4 号様式：貨物自動車運送事業事業実績報告書）	15	130.4	1
1203	貨物自動車運動事業報告規則に基づく事業報告書(1 号様式、2 号様式、3 号様式、貸借対照表、損益計算書）	106	341.6	11.1
1703	内航海運事業に係る報告書（内航海運業事業概要報告書）【紙】	22	244.8	4
1707	「内航船省エネルギー格付制度」事務取扱要領様式第 1、第 2、第 3、第 4	21	79.9	1
1708	船員法第 111 条に基づく事業状況に関する報告	50	173.2	1
1901	倉庫業登録申請書（倉庫業登録申請書）	46	224.7	33.14
1911	【様式第 1 号】総合効率化計画認定申請書	30	1,023.0	29.95
1912	【様式第 6 号】流通業務総合効率化事業実施状況報告書	63	257.4	4.03
1913	【第 1 号様式】新・増設倉庫証明申請書	39	156.7	3.92
2001	自動車事故報告書別記様式（第 3 条関係）	59	313.6	3.77
2501	自家用自動車有償貸渡事業 貸渡実績報告書（レンタカー事業） 【様式 1】貸渡実績報告書	25	106.2	1
2502	自家用自動車有償貸渡事業 貸渡実績報告書（レンタカー事業） 【様式 2】事務所別車種別配車両数一覧	20	50.6	1

2503	レンタカー協会 PDF	14	134.5	1
3301	運転事故等届出書（第 2 号様式）または 運転事故等報告書（第 1 号様式）	49	219.5	8.55
3302	災害報告書（第 6 号様式）	20	175.1	7.19

さらなる指標として単位時間あたりの処理件数を示すスループットを計測した。本検証では、単位時間を 10 分とした。計測には 2501「貸渡実績報告書」を対象にした。同時実行パターンを変えながら計測を行った結果、処理時間と 1 件あたり処理時間は以下のスループット結果の表の通りとなった。

上記のスループット検証結果から、得られた重要な知見としては、まず、同時実行数とファイル数の総リクエスト数が増加しても、スループット （10 分あたりの処理件数） は約 1250~1678 件の範囲内に収まっており、システムのスケーラビリティが確認できたことである。特に、最大負荷である 2,500 リクエスト（50 ユーザー×50 ファイル）においても、10 分あたり 1678 件という最も効率的な処理時間を記録している点は特筆すべき結果である（表 39）。これは、並列処理の効率が高負荷時により発揮されることを示唆している。

また、総リクエスト数が少ない場合（350〜500 リクエスト）では、10 分あたり 1313 件とややとやや少なくなっている。これは、並列処理のウォームアップ時間や、リソースの初期割り当てに伴うオーバーヘッドが、相対的に大きく影響するためと考えられる。実運用においては、ある程度まとまった件数をバッチ処理することで、より高い処理効率が期待できる。ただし、実際の運用では前後処理（ファイルアップロード、結果のダウンロード、エラーハンドリング等）や、ネットワーク遅延、一時的な負荷集中などを考慮する必要がある。

表 4.30　スループット結果

同時実行パターン（ファイル数）	処理時間	スループット （10 分あたりの処理件数）
50 sessions x 50 files = 2,500 requests	894s (~15min)	1678
50 sessions x 30 files = 1,500 requests	566s (~9min)	1590
35 sessions x 50 files = 1,750 requests	618s (~10min)	1699
35 sessions x 30 files = 1,050 requests	435s (~8min)	1448
50 sessions x 10 files = 500 requests	240s (~4min)	1250
35 sessions x 10 files = 350 requests	160s (~3min)	1313

4.10　課題と今後の取り組み

本章の検証から、今年度は「OCR＋LLM（Azure OCR + Claude Sonnet）」を標準構成として採用しつつ、丸囲み・打ち消し線など視覚的解釈が必要な箇所は検知と切替により VLM を併用する方針が妥当であることを確認した。一方で、実運用・大規模展開を見据えると、精度・処理性能・運用設計の各観点で残る課題がある。

4.10.1　残課題

●　視覚認識が伴う課題（丸囲み・打ち消し線・図形）に対する汎用性の担保

特に打ち消し線検知は各業務ごとの書類データ間で分布差が大きく、条件が変わると精度が低下し得るため、汎用的に効く特徴量および学習データ設計が必要となる。

●　VLM 併用時の“切替ルール”と“上書き範囲”の標準化

どの特徴で OCR＋LLM／VLM 併用に切り替えるか、また VLM でどの項目をどこまで上書きするかが、現時点では評価条件依存になりやすい。誤検知（FP）・見逃し（FN）時の再処理方針や、品質担保のためのガードレール（例：上書き対象の限定、矛盾検知）を整理する必要がある。

●　OCR 依存の精度・処理時間のボトルネック

文章抜き出しは OCR 品質に強く依存し、入力品質が悪い場合の下振れが残る。OCR 工程がスループット上の支配要因になりやすく、並列化・キャッシュ・再処理単位の最適化を含めた処理設計の詰めが必要。

●　評価設計（KPI・再現性）のさらなる整備

精度（全カラム平均、文章／丸囲みの内訳）に加え、処理時間・コスト・運用性を含む評価を継続し、改善がどの要因に起因するかを追跡できる形にする必要がある。多様な文書に対応するには、評価項目においても同様に様式、画質、手書き量、図形密度など多様な項目を対象として、評価することが望ましい。

●　OSS 活用（VLM/LLM、ファインチューニング）の運用負荷

LoRA 等で精度改善の可能性はある一方で、学習データ整備、モデル管理、推論基盤運用、再学習まで含めた運用負荷が大きく、採用判断の材料が不足している。

4.10.2　次年度以降の取り組み

●　文書特性に応じた推論ルート自動選定の実装・高度化

「精度・速度・コスト」を同時に満たすため、文書の特徴量に基づき OCR＋LLM／VLM 併用を切り替える判定ルールを明文化し、再現性のある手順として実装する。低コストモデル（例：Haiku）活用を含め、選定結果と品質の関係を継続的に計測・更新する。

●　打ち消し線・選択式・図形の“検知→抽出→統合”パイプラインの標準化

打ち消し線検知は学習データ拡充と層別評価を行い、様々なデータに対して情報抽出ができるように汎用性を高める。VLM 上書きの対象項目・ページ特定・ページ番号突合など、統合手順を実装に落とし込み、再処理範囲を最小化しつつ品質を上げる。

● **KPI（精度・性能・コスト・運用性）の継続計測と改善サイクルの確立**

共通評価軸（精度・速度・コスト・運用性・拡張性）に沿った計測を定常化し、改善施策の効果を継続的に検証する。代表的な業務の書類だけでなく、難易度の異なる帳票群を含めたベンチマークセットを整備し、標準構成の妥当性を継続的に再評価する。

第 5 章　データ活用モジュール（LINKS BI）における技術的工夫と性能検証

データ活用モジュール（LINKS BI）は、非エンジニアである国交省職員が行政データの分析・可視化を自律的に行うことができる 環境を提供することを目的とするサブシステムである。2024 年度にはユースケースごとに個別のアプリケーションを開発する運用をとっていたが、操作の習得コストの高さや大規模データの処理性能に課題があった（詳細は 5.6 節参照）。2025 年度ではこれらの課題を解決するために、AI チャットによる自然言語操作と DuckDB-Wasm によるブラウザ内データ処理を中核とする新しい LINKS BI をゼロから設計・実装した。

本章では、この構築にあたって採用した技術的工夫と、AI チャット機能の精度・性能に関する検証結果を記述する。前半（5.1 節～5.6 節）では、DuckDB-Wasm によるブラウザ内分析基盤、エージェンティック AI アーキテクチャ、スキルシステム、可視化技術、国会答弁案生成の最適化、UI/UX 改善といった技術的工夫を解説する。後半（5.7 節）ではこれらの工夫に基づく AI チャット機能の精度検証と処理性能検証の結果を示し、最後に得られた知見と今後の方針（5.8 節～5.9 節）を述べる。

5.1　ブラウザ内データ分析基盤（DuckDB-Wasm）

5.1.1　採用の背景：大規模データへの対応

LINKS BI では大規模なデータセットを扱うユースケースが想定されている。たとえばとあるモビリティの運行計画に関するデータは約 100 万～200 万レコード規模に及ぶ。このようなデータに対して集計・統計処理・可視化を行う場合、通常のリレーショナルデータベースではスケーラビリティに問題が生じやすく、クエリのレスポンスにも大きな時間がかかることがある。

OLTP と OLAP：データベースの 2 つの処理モデル

データベースの処理モデルは、大きく OLTP（Online Transaction Processing、オンライントランザクション処理）と OLAP（Online Analytical Processing、オンライン分析処理）の 2 つに分類される。

OLTP は、個々のレコードの挿入・更新・削除・参照といったトランザクション処理に最適化されたモデルである。業務システムにおけるデータの登録・編集・検索など、少量のレコードに対する高頻度な読み書き操作を高速に処理することを目的としている。PostgreSQL・MySQL などの一般的なリレーショナルデータベースは OLTP 向けに設計されている。

OLAP は、大量のデータに対する集計・統計・分析クエリに最適化されたモデルである。「全レコードから特定カラムの合計値を算出する」「複数の条件でグループ化して集計する」といった、広範囲のデータを横断的に読み取る処理を高速に実行することを目的としている。列指

向のデータ格納方式（本節参照）と組み合わせることで、分析クエリに対して高い I/O 効率を実現できる。
本システムでは、データ構築モジュールの Silver 層のデータ管理には OLTP 型の PostgreSQL を、Gold 層のデータ変換・分析には OLAP 型の DuckDB を採用しており、処理特性に応じてデータベースを使い分けている。
こうした分析用途に特化した OLAP データベースとして、DuckDB を採用した。DuckDB は、オープンソースの OLAP データベース管理システムであり、SQL を用いて CSV・JSON・Parquet などのファイルデータをインメモリで高速に処理できる。列指向の特性から、特定カラムの集計・フィルタリング処理では行指向の OLTP データベースと比較して大幅な高速化が期待できる。
DuckDB はデータ構築モジュールのサーバーサイドでもデータ変換・集計処理に使用しているが、LINKS BI では後述する DuckDB-Wasm という形でブラウザサイドにも展開している。

5.1.2　DuckDB-Wasm とブラウザ内処理

WebAssembly（WASM）とは

WebAssembly（WASM）とは、モダンなブラウザに搭載されているバイナリ形式の実行環境である。JavaScript を使わず C・C++・Rust などで書かれたプログラムをブラウザ上で動作させることができる。本来はサーバーサイドや専用環境での動作を想定して書かれたライブラリも、WebAssembly 形式にコンパイルされることでブラウザ上にロードされ、JavaScript から呼び出して実行できるようになる。
DuckDB-Wasm はこの仕組みを利用して、C++で実装された DuckDB エンジンをブラウザ上で動作可能にしたものである（DuckDB-Wasm 公式リポジトリ）。サーバーにクエリを送信することなく、ブラウザ内で SQL による高速なデータ分析が実行できる。
DuckDB-Wasm の採用を決定した背景には、前年度（2024 年度）の実装における課題がある。2024 年度の Veda では、データの配信形式として GeoJSON および GeoJSONL（ストリーミング処理に対応した行区切りの GeoJSON 形式）を採用していた。2024 年度においても GeoJSONL を gzip 圧縮することで転送量削減の対策を講じていたが、100 万行規模のデータに対してはそれでも十分ではなく、テキストベースのフォーマット固有の限界が残っていた。
さらに、取得したデータに対する集計・フィルタリング等の処理もフロントエンド上の独自実装で行っていたため、大規模データを扱う際にブラウザのメインスレッドへの処理負荷が極めて高くなっていた。その結果、画面がフリーズして操作を受け付けなくなる、あるいはページ自体がまともに開けないといった状況が頻繁に発生しており、実用上の大きな障壁となっていた。
こうした課題を解消するために、2025 年度の LINKS BI では DuckDB-Wasm を採用し、データ分析処理をブラウザ内で完結させる設計に刷新した。DuckDB-Wasm と Parquet の組み合わせにより、前年度の各課題に対して以下の効果が得られた。

- 第 1 に、データ転送コストの大幅な削減である。GeoJSON/GeoJSONL はテキストベースのフォーマットであり、数値やジオメトリ情報が JSON 文字列として冗長に表現されるため、100 万行規模のデータでは転送サイズが膨大になっていた。Parquet は列指向のバイナリフォーマットであり、同じ型・近い値のデータが連続する構造を利用した高効率な圧縮が可能である。さらに、DuckDB-Wasm は Parquet のメタデータを利用して必要な列・行グループだけを部分的に取得できるため、クエリに関係しないデータはそもそもダウンロードされない。これにより、ネットワーク転送量が大幅に削減された。
- 第 2 に、分析処理速度の根本的な改善である。前年度の独自実装では JavaScript で行単位にデータを走査して集計処理を行っていたため、大規模データに対して処理効率が著しく低下していた。DuckDB-Wasm は列指向の OLAP エンジンであり、集計・フィルタリング・グループ化といった分析クエリを SQL で記述するだけで、列指向に最適化されたエンジンが高速に処理する。行指向の独自実装と比較して、分析処理の実行速度が格段に向上した。
- 第 3 に、スケーラブルなアーキテクチャの実現である。Parquet ファイルをブラウザが GCS から直接取得し、DuckDB-Wasm がブラウザ内で SQL クエリを実行する構成により、分析処理の負荷はすべて各ユーザーのブラウザ上にかかる。サーバー側にはデータ集計や配信の負荷が発生しないため、ユーザー数が増加してもサーバーのスケールアップが不要であり、サーバーに依存しないスケーラブルな構成が実現された。
- 第 4 に、インフラ運用コストの削減である。従来の RDBMS サーバー（PostgreSQL 等）を BI 用に維持する必要がなくなり、インフラの運用管理コストが大幅に削減された。DuckDB-Wasm が内部では Apache Arrow の列指向メモリレイアウトをネイティブに使用しているため、Parquet から Arrow への変換オーバーヘッドなしにクエリ結果を Vega-Lite・MapLibre GL JS へ高速に受け渡せる。

5.2 AI チャット機能のアーキテクチャ

LINKS BI の AI チャット機能は、エージェンティック AI 技術を基盤として設計されている。エージェンティック AI（Agentic AI）とは、与えられた目標に対して AI が自ら計画を立て、外部ツールを選択・実行しながら能動的にタスクを達成する AI システムの総称である。Anthropic 社はエージェンティックシステムを「LLM が自身の処理とツール使用を動的に指示し、タスクの達成方法を自ら制御するシステム」と位置づけている（参考: Anthropic - Building Effective Agents）。

従来の生成 AI との違いを具体例で示す。たとえば「今日の天気は？」と質問した場合、従来の生成 AI は学習データに基づいて「おそらく晴れだと思います」のようにテキストを返すだけであった。一方、エージェンティック AI は、受けた質問に対し、天気予報 API で最新の気象情報を取得し、位置情報 API で現在地を特定した上で「東京都千代田区の現在の天気は晴れ、最高気温は 25 度です」のようにリアルタイムの事実に基づいた回答を返す。AI がどのツールをどの順番で使うかを自ら判断し、複数のステップを自律的に実行する点が本質的な違いである。

この動作を実現する中核技術が Function calling（5.2.1 項参照）であり、LINKS BI の AI チャット機能はこのエージェンティックな動作パターンに基づいて設計されている。

5.2.1　Function calling とは

LLM を活用して AI チャット機能を実現する際、単にテキストを生成させるだけでは実際のデータ操作や可視化を行うことができない。この問題を解決する仕組みが Function calling である。なお、Function calling は Tool use（ツール使用）とも呼ばれ、AWS Bedrock の公式ドキュメントでは「Tool use (function calling)」として両方の用語が併記されている（参考: AWS Bedrock - Tool use）。

Function calling（関数呼び出し）は、OpenAI 社が 2023 年 6 月に GPT-4・GPT-3.5 Turbo への機能追加として初めて導入した技術である（参考: OpenAI Function calling ガイド）。LLM はテキスト生成に特化しており、学習時点以降の最新データへのアクセスや外部システムとの連携が原理的にできない。Function calling はこのギャップを埋めるために設計された仕組みであり、「モデルに外部システムとのインターフェースを提供する方法」と OpenAI の公式ドキュメントで定義されている。

OpenAI のドキュメントが示す動作フローは以下の 5 ステップで構成される。

1. ツール定義の提供：アプリケーションがツールの名前・説明・引数の JSON Schema をシステムプロンプトとともにモデルに送信する。
2. モデルによる判断：モデルは会話の文脈からどのツールをどの引数で呼び出すべきかを判断し、ツール呼び出し命令を含むレスポンスを返す。
3. アプリケーションによる実行：アプリケーション側がその命令を受け取り、実際の処理（データベースクエリ・API 呼び出し等）を実行する。
4. 結果の返却：実行結果をツール実行結果メッセージとしてモデルに送り返す。
5. 最終応答の生成：モデルはツール実行結果をもとに次の判断や応答生成を行う。

このループを繰り返すことで、AI が自律的に複数のツールを組み合わせながら複雑なタスクを達成するエージェンティックな動作が実現される。Function calling は OpenAI の GPT-4 を皮切りに、Anthropic 社の Claude・Google 社の Gemini など主要な LLM プロバイダーが相次いで対応し、現在では生成 AI アプリケーション開発における標準的な機能となっている。

5.2.2　LINKS BI における Function calling の実装

LINKS BI の AI チャット機能は、AWS Bedrock の Claude モデルと Function calling 機構を組み合わせて実装されている。ユーザーが自然言語で指示を入力すると、バックエンドの API サーバーが Bedrock へストリーミングリクエストを送信する。Claude は BI アプリケーションに定義されたツール群の中から適切なものを選択し、ツール呼び出し命令を Vercel AI SDK 独自の HTTP ストリーミング形式でブラウザに配信する。ブラウザがこの命令を受け取って実際のツール処理を実行し、結果を次のリクエストとして API サーバー経由で Claude に返す。この Function calling のループが完了するまで繰り返され、最終的な回答がユーザーに表示される。

定義されているツールの主な例を以下に示す。

- DuckDB SQL クエリの実行：ブラウザ内の DuckDB-Wasm に対して SQL を発行し、集計・フィルタリング・統計計算等を行う。
- グラフの可視化：Vega-Lite の仕様に従ったチャート定義を生成し、画面にグラフを描画する。
- 地図表示：MapLibre GL JS を用いて緯度経度データをマップ上にプロットする。
- 回帰分析・クラスタ分析：ブラウザ内の分析ライブラリを呼び出して統計処理を実行する。

本システムの Function calling の特徴は生成と実行の分離にある。Claude はクラウド上で SQL 文・グラフ仕様・地図スタイルを生成するが、実際の処理（SQL 実行・グラフ描画・地図描画）はすべてブラウザ内で実行される。以下の概念図にこの実行アーキテクチャを示す。

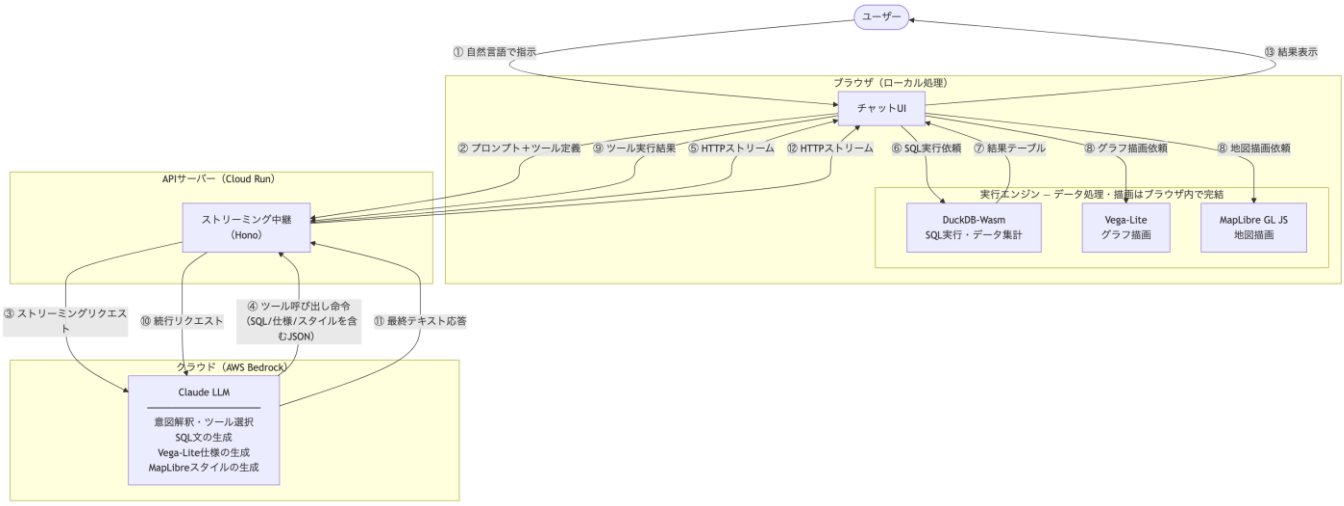


図 39 LINKS BI AI チャット — Function calling の実行アーキテクチャ概念図

図の読み方：ステップ(1)〜(5)で自然言語のプロンプトが Claude へ届き、Claude がツールの種別と引数（SQL 文・Vega-Lite 仕様・MapLibre スタイル）を生成してブラウザへ返す（(4)）。ステップ(6)〜(8)でブラウザ内の各エンジン（DuckDB-Wasm・Vega-Lite・MapLibre GL JS）が実際の処理を実行する。ステップ(9)〜(10)でその結果を Claude へフィードバックし、必要に応じてループを繰り返す。このサイクルにより、サーバーに大規模なデータを転送することなく、ブラウザ内で完結した高速な AI 分析が実現される。

Server-Sent Events（SSE）とは

SSE は、サーバーからクライアント（ブラウザ）へ一方向のリアルタイムデータ配信を実現する HTTP 標準技術である。サーバーが 1 つの HTTP コネクションを維持したまま、データが生成されるたびにクライアントへ逐次送信する仕組みであり、HTML5 仕様の一部として W3C で標準化されている。

類似技術として WebSocket がある。WebSocket はサーバー・クライアント間の双方向通信を実現するプロトコルであり、チャットアプリやリアルタイムゲームなど双方向のやり取りが頻繁に発生する用途に適している。一方、SSE はサーバーからの一方向配信に特化しており、通常の HTTP 上で動作するため実装が簡素で、ロードバランサーやプロキシとの互換性も高い。LLM を用いた AI チャットで SSE が広く採用される理由は、LLM の応答生成がトークン単位の逐次処理であることに起因する。LLM はプロンプトを受け取ると応答テキストを一度に生成するのではなく、トークン（単語や文字の断片）を 1 つずつ順番に生成する。SSE を用いると、生成されたトークンをリアルタイムにブラウザへ配信できるため、ユーザーは応答全体の生成完了を待たずに途中経過を確認できる。このストリーミング配信はサーバーからクライアントへの一方向で十分であり、WebSocket の双方向通信は不要である。OpenAI・Anthropic・AWS Bedrock など主要な LLM API プロバイダーもストリーミングレスポンスの配信方式として SSE を採用している。なお、LINKS BI では Vercel AI SDK を利用しており、クライアント・サーバー間の通信には SSE ではなく ai-sdk 独自の HTTP ストリーミング形式を使用している。

5.2.3　Vega-Lite の採用と AI との相性

グラフ可視化には Vega-Lite を採用している。Vega-Lite は、ワシントン大学 Interactive Data Lab（UW IDL）が開発するオープンソースの可視化ライブラリであり、公式サイトでは「インタラクティブグラフィクスのための高水準グラマー（A High-Level Grammar of Interactive Graphics）」と定義されている。Vega-Lite は上位プロジェクトである Vega（宣言的な可視化グラマー）の上に構築されており、JSON で記述した簡潔な仕様が Vega 仕様へと自動コンパイルされる。棒グラフ・折れ線グラフ・散布図・地図等、多様なチャートタイプを統一的な JSON 形式で定義できる。

Vega-Lite の設計思想は、Leland Wilkinson が提唱した「Grammar of Graphics（グラフィクスの文法）」に基づいている。これは、グラフを「棒グラフ」「折れ線グラフ」といった固定的なチャートタイプとして扱うのではなく、データ・エンコーディング（視覚的属性への対応づけ）・マーク（点・線・矩形等の図形要素）・スケール・座標系といった独立した構成要素の組み合わせとして記述する考え方である。R 言語の ggplot2 も同じ思想に基づいており、Vega-Lite 公式サイトでは ggplot2 や Tableau と同等のアプローチであると説明されている。

Vega-Lite を採用した主な理由は AI との親和性である。グラフ可視化ライブラリの実装方式は、大きく手続き的（命令型）アプローチと宣言的アプローチに分類できる。Chart.js・D3.js・ECharts 等の手続き的ライブラリでは、可視化の定義を JavaScript コードとして記述する必要がある。キャンバス要素の取得、データ配列の構築、スケール関数の設定、軸の描画、イベントリスナーの登録といった一連の処理を、開発者が JavaScript の文法に従ってプログラムとして記述する。これはプログラミング言語の実行環境に依存する「コード」である。

一方、Vega-Lite では可視化の定義を JavaScript コードではなく、純粋な JSON データ構造として記述する。以下は Vega-Lite 公式サイト（Stacked Bar Chart with Rounded Corners）に掲載されているシアトルの天気データの積み上げ棒グラフの仕様である。

```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v6.json",
  "data": {"url": "data/seattle-weather.csv"},
  "mark": {"type": "bar", "cornerRadiusTopLeft": 3, "cornerRadiusTopRight": 3},
  "encoding": {
    "x": {"timeUnit": "month", "field": "date", "type": "ordinal"},
    "y": {"aggregate": "count"},
    "color": {"field": "weather"}
  }
}
```

この JSON 仕様を Vega-Lite のレンダラーに渡すと、以下のような積み上げ棒グラフが自動生成される。

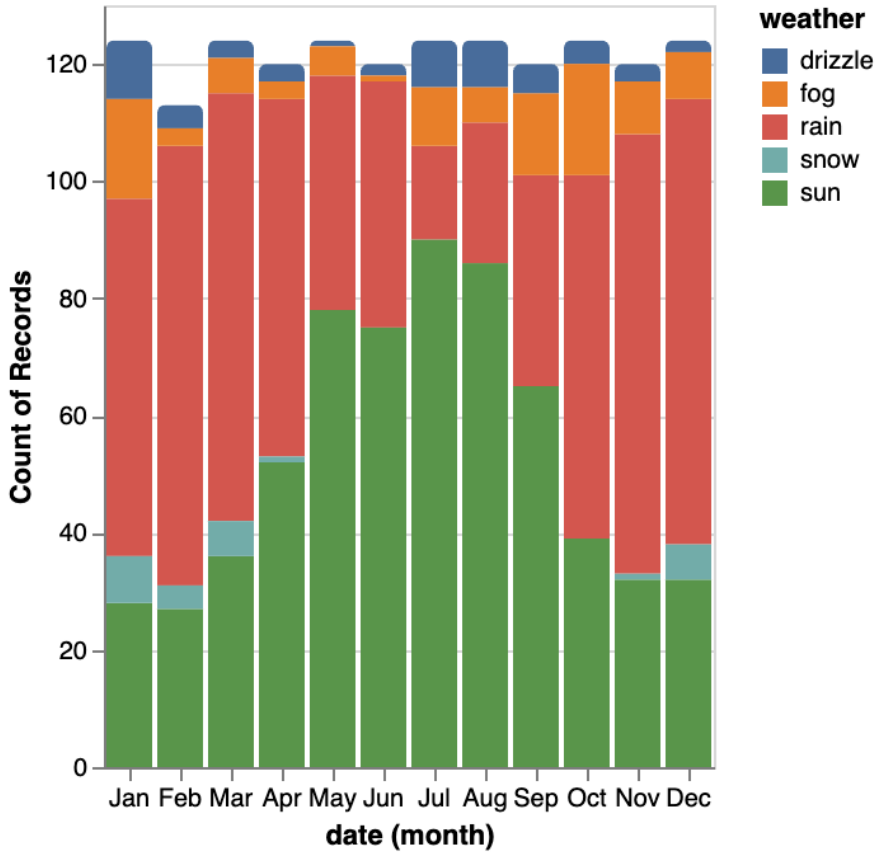


図 40 Vega-Lite 公式サイトのシアトル天気データの例（月別の天気種別の積み上げ棒グラフ）

注目すべき点は、この仕様の中に JavaScript のコードが一切含まれていないことである。データソース（「data」）・マークの種類（「mark」）・エンコーディング（「encoding」：どのデータフィールドをどの視覚属性に対応づけるか）を宣言するだけで、軸・スケール・凡例・色分け・レイアウトはすべて Vega-Lite が自動生成する。

この違いが AI によるグラフ生成において決定的な意味を持つ。Vega-Lite の仕様構造は JSON Schema として公式に定義・公開されている。JSON Schema とは、JSON データがどのようなフィールド・型・値を持つべきかを形式的に記述する仕様である。つまり LLM にとっては、このスキーマ構造に適合する形で Vega-Lite の設定 JSON を生成すればよく、生成結果が正しいかどうかをシステム側で JSON Schema バリデーションによって機械的に検証できる。手続き的ライブラリの場合、LLM が生成した JavaScript コードが正しく動作するかを検証するにはコードを実際に実行するしかなく、構文エラー・ランタイムエラー・DOM 操作の誤りなど多様な障害点が存在する。

加えて、JSON 仕様はデータであってコードではないため、LLM が生成した仕様をブラウザ内で処理してもスクリプトインジェクション等のセキュリティリスクが生じない点も重要な利点である。

5.2.4　プロンプトエンジニアリングによる動作制御

AI が意図通りに動作するためには、Function calling の定義に加えてプロンプトエンジニアリングによる細かな制御が不可欠である。LINKS BI では以下のような工夫を実施している。

第 1 の工夫はワンショット実行の強制である。LLM は途中でテキストを返して処理を止める傾向があるが、国会答弁案生成などの複数ステップを要する処理では「全ステップを 1 レスポンスで完了せよ。途中で止まるな」という強い制約をシステムプロンプトに明示し、AI が自律的に最後まで処理を完了するよう誘導している。

第 2 の工夫は構造化出力の強制である。チャットの最終応答をシステムが機械的に扱えるよう、特定の XML タグ（例：「<!--SUMMARY-->〜<!--/SUMMARY-->」）で囲んだ形式での出力をシステムプロンプトで強制している。これにより、AI の応答から必要な情報を確実に抽出して UI に反映できる。

第 3 の工夫はバリデーションループの組み込みである。答弁案生成では、生成した答弁案の文字数・段落数をツールで検証し、基準を満たさない場合は AI に修正を指示して再生成させるループを設けている。AI が自律的に品質基準を満たすまで繰り返す設計であり、最大試行回数に達した場合は強制的に完了として扱う。

第 4 の工夫は強調キーワードの階層構造である。システムプロンプト内で AI に指示の優先順位を正しく判断させるために、CRITICAL・MUST・NEVER などの強調キーワードに以下の使用制限を設けている。

表 5.1　強調キーワードの使用制限

レイヤー	許可キーワード	禁止キーワード
RULE ZERO （システムプロンプト）	CRITICAL, MUST, NEVER	なし
スキルファイル （5.3 節で後述）	IMPORTANT, MUST, NEVER, MANDATORY	CRITICAL
ツール description	MUST, NEVER	CRITICAL, IMPORTANT

CRITICAL をスキルファイルで使用すると階層が崩壊し、AI が最優先の指示を識別できなくなる。そのため、CRITICAL は RULE ZERO（ツール実行パターンの根幹ルール）専用として厳格に運用している。

第 5 の工夫は temperature パラメータの固定である。行政業務で使用するシステムでは応答品質の安定性が不可欠であるため、temperature パラメータを 0 に設定し、同一入力に対する結果のブレを最小化している。

temperature パラメータとは

LLM の出力のランダム性（多様性）を制御するパラメータである。0 に近いほど出力が安定・決定論的になり（同一入力に対して同一の出力に収束しやすい）、1 以上に近づくほど多様でランダムな出力が生成される。

5.3　スキルシステムによるプロンプト管理

5.3.1　AI チャットにおけるプロンプト管理の課題

LINKS BI の AI チャット機能は、データ結合・グラフ可視化・地図表示・統計分析・国会答弁案生成・経路探索など多様なユースケースに対応する必要がある。これらのユースケースごとに AI への指示（プロンプト）を作成すると、すべての指示をシステムプロンプトに集約する設計ではプロンプトが数百〜数千行に膨張し、以下の問題が発生する。

- AI が指示を遵守しなくなる：プロンプトが長大になると AI の注意力が分散し、重要な指示を無視する頻度が増加する。最も深刻なケースでは、AI がツールを一切呼び出さずにデータを参照せずに回答を生成し、事実に基づかない内容を出力するハルシネーションが発生した。
- コンテキストウィンドウの圧迫：すべての指示が常時読み込まれるため、実際に必要な指示以外のテキストがトークンを消費し、AI の応答品質が低下する。
- 指示間の競合：異なるユースケースの指示が同一のプロンプト内に混在することで、矛盾や優先順位の曖昧さが生じる。
- 保守性の低下：プロンプトの一部を変更した際の影響範囲が広く、回帰テストのコストが増大する。

コンテキストウィンドウとは

コンテキストウィンドウ（Context Window）とは、LLM が 1 回のやり取りで参照できるテキストの最大量を指す。LLM はこのウィンドウ内に含まれる情報のみを「記憶」して回答を生成するため、ウィンドウに入りきらない情報は参照できない。たとえば Claude Sonnet 4.5 のコンテキストウィンドウは 200,000 トークン（日本語で約 15 万文字相当）である。一見十分に大きいが、システムプロンプト・会話履歴・ツール定義・スキルの内容がすべてこの枠を共有するため、エージェンティック AI のように複雑な処理を行うシステムではコンテキストウィンドウの効率的な使用が性能に直結する。コンテキストウィンドウの上限に近づく、もしくは超えると、AI がそれ以前の会話の内容を考慮できなくなり、いわゆる指示を忘れる状態が発生する。そのため、必要な情報だけを必要なタイミングで AI が読み込む設計が重要になる。

5.3.2　段階的開示によるコンテキスト最適化

上記の課題に対して、LINKS BI では AI への指示内容をシステムプロンプト本体に集中させるのではなく、スキルファイルに分散配置する「Skills-First」アーキテクチャを採用している。この設計方針では、システムプロンプト本体は約 100 行の最小限の記述にとどめ、すべての詳細な指示はスキルの Markdown ファイルに記述する。

この設計の基盤となっているのが、Anthropic 社が提唱する Agent Skills（エージェントスキル）の概念である。Agent Skills とは、AI エージェントの汎用的な能力を特定のドメインに特化させるための拡張モジュールであり、「インストラクション・メタデータ・オプションのリソース（スクリプト・テンプレート等）をパッケージ化し、関連するタスクにおいて AI が自動的に使用する再利用可能な機能単位」と定義されている（参考: Anthropic - Agent Skills Overview）。

Agent Skills の設計原理の中核は段階的開示（Progressive Disclosure）にある。Anthropic 社のエンジニアリングブログでは、スキルを「目次から始まり、各章へ、そして詳細な付録へと進む、よく構成されたマニュアル」に例えている（参考: Anthropic - Equipping Agents for the Real World with Agent Skills）。具体的には、情報を以下の 3 段階で読み込む設計になっている。

1. メタデータ（常時読み込み）：スキルの名前と説明をシステムプロンプトに含め、AI がどのスキルが利用可能かを常に把握する。1 スキルあたり約 100 トークンと軽量であるため、多数のスキルを登録してもコンテキストウィンドウを圧迫しない。
2. インストラクション（トリガー時に読み込み）：ユーザーの要求がスキルに合致した場合にのみ、詳細な手順・制約・出力形式を記述したスキルファイルを読み込む。
3. リソース（必要時に読み込み）：スクリプト・テンプレート・参照資料等の補足ファイルは、インストラクションから参照された時点で初めて読み込まれる。

これらのポイントは、AI にどのようなスキルが存在するかを極力少ないトークンで把握させ、AI が必要と判断したときにそのスキルのより詳細な指示を読み込むことで、コンテキストウィンドウの効率的な使用を実現する点にある。

以下のシーケンス図に、この段階的開示の流れを示す。

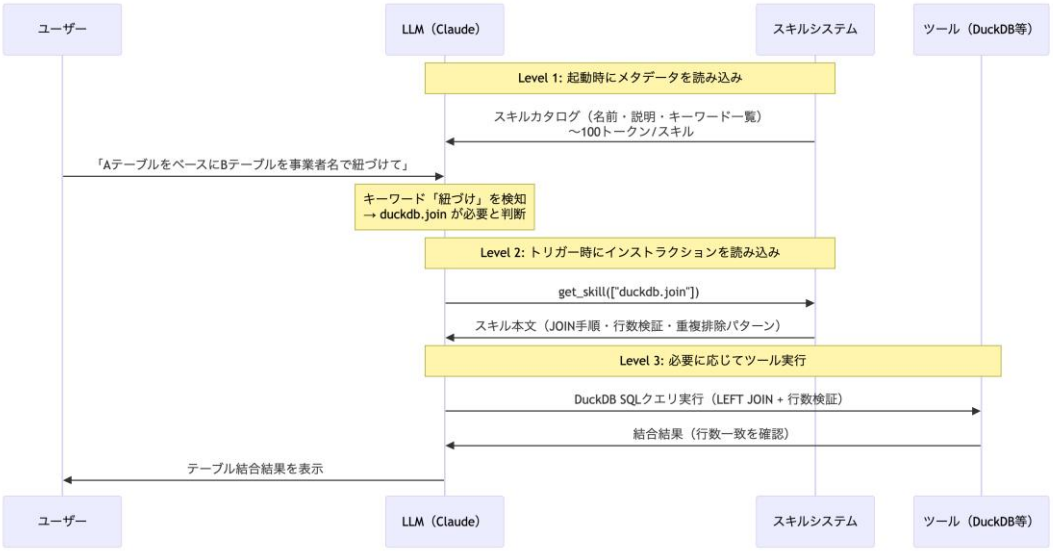


図 41 スキルの段階的開示（Progressive Disclosure）の流れ

この段階的開示により、必要な情報だけがコンテキストウィンドウに読み込まれ、不要な情報がトークンを消費しない効率的なアーキテクチャが実現される。通常のプロンプト（会話レベルの一回限りの指示）とは異なり、スキルはオンデマンドで読み込まれる再利用可能なパッケージであり、同じ指示を繰り返し提供する必要がなくなる。

5.3.3 LINKS BI におけるスキルの実装

LINKS BI では、Anthropic 社の Agent Skills が提唱するファイルシステムベースのアーキテクチャを踏襲し、スキルファイルの追加・修正が即座に AI の動作に反映される仕組みを実装している。

スキルファイルは Markdown 形式で 「skills/{ドメイン}/{スキル ID}.md」 というパスに配置される。ビルド時に Vite の 「import.meta.glob」 機能によってこれらのファイルが自動的に発見され、ファイルパスからスキル ID が生成される（例：「skills/duckdb/fiscal-year.md」 → 「duckdb.fiscal-year」）。このため、スキルファイルを所定のディレクトリに追加するだけで、新しいスキルが AI から利用可能な状態になる。

各スキルファイルのフロントマター（YAML メタデータ）には以下の 3 つのフィールドを定義する。

- 「description」（必須）：スキルカタログに表示される 1 行の説明。AI がどのスキルを取得すべきかを判断するための情報である。
- 「tasks」（任意）：ルーティングキーワード。ユーザーのメッセージに含まれるキーワードとマッチングし、対応するスキルの取得を AI に促す。日本語・英語の両方のキーワードを推奨する。

- 「deps」（任意）：依存スキル ID。あるスキルが別のスキルの知識を前提とする場合に記述し、自動的に co-fetch（同時取得）される。依存スキルは常に本体スキルより先に取得される順序が保証されており、前提知識が確立された上でスキルの指示が読み込まれる。

各スキルファイルは 200 行以下に保つことが推奨される。200 行を超えるスキルは AI の注意力を希釈し、トークンコストも増大するため、同一ドメインディレクトリ内のサブスキルに分割する。

ツールとスキルの関係は Prerequisite（前提条件）として明示的に定義されている。たとえばグラフ生成ツール（「update_vega_graph_spec_for_table」）は 「vega」 ドメインのスキルを事前に取得していない限り実行を拒否する。AI がスキル取得をスキップしてツールを直接呼び出そうとした場合、ツールはエラーを返して 「get_skill」 の呼び出しを促す。この仕組みにより、AI が必要な実装指針を把握せずにツールを誤用するリスクを排除している。

もう 1 つの重要な設計原則は、ドメイン規約と詳細指示の分離である。AI がスキルを取得するかどうかを判断するためには、スキル取得前の時点で一定の文脈知識が必要になる場合がある。たとえば日本の行政統計における「年齢別」という用語は慣例的に 10 歳刻みの年代グループ化を意味するが、この知識がなければ AI は「年齢別」を文字通りに解釈し、「duckdb.age-grouping」 スキルの取得をスキップしてしまう。このような文化・分野固有の暗黙知は 1〜2 行のドメイン規約としてシステムプロンプトに記載し、具体的な実装手順（CASE WHEN 文の SQL パターン等）はスキルファイルに記載するという分離を行っている。

5.3.4　スキルの具体例

LINKS BI では、汎用チャット機能を基盤としつつユースケースに特化したスキルを追加できるプラグインアーキテクチャを採用している。各スキルは以下の要素で構成される。

- システムプロンプトセクション：そのスキルで AI が従うべき処理手順・制約・出力形式を定義する。
- ツール群：スキル固有のデータ操作・外部 API 呼び出し・バリデーション等を実装した Function calling 用ツールである。
- UI パネル：スキルに特化したインタラクション UI（フォーム・選択肢等）を提供する。

スキルの具体的なイメージを示すため、実際に実装されている「テーブル結合スキル」（「duckdb.join」）の内容をダイジェストで紹介する。このスキルは、2 つのテーブルを JOIN で結合する際の手順・検証パターン・重複排除ルールを AI に教えるものである。スキルファイルの冒頭には以下のようなフロントマター（メタデータ）とインストラクション（本文）が記述されている。

```
---
description: JOIN operation verification patterns, row count validation,
  and join key selection rules
```

```
tasks: JOIN, 結合, 紐づけ, マージ
---

## JOIN Operations

ユーザーが指定した結合キーのみを使用し、勝手に追加の条件を付与しない。
「A をベースに」と指示された場合、結合後の行数がベーステーブルと
一致することを必ず検証し、増加した場合は ROW_NUMBER()で重複排除する。
...
```

「description」 はスキルカタログに表示される 1 行の説明であり、AI がどのスキルを取得すべきかを判断するための情報である。「tasks」 にはルーティングキーワードが定義されており、これらのキーワードをもとに逆引き索引が生成されて AI のツール説明として組み込まれる。これにより AI が「結合」「紐づけ」等のキーワードを含む指示を受けた際に関連するスキルを選択しやすくなっている。

フロントマターに続くインストラクション（本文）には、結合キーの選択ルール・行数検証の手順・重複排除の SQL パターン（「ROW_NUMBER()」による 1:N 結合の解消）が記述されている。AI はこのスキルを取得した時点で JOIN 操作の正しい手順を把握し、ユーザーが「A テーブルをベースに B テーブルを紐づけて」と指示するだけで、行数整合性を保証した DuckDB SQL クエリを生成できるようになる。

このように、スキルは「ドメイン固有の知識」と「それを実行するための具体的な手順」を Markdown ファイル 1 つにパッケージ化したものであり、スキルファイルを追加するだけで AI の能力を拡張できる。以下に、現在実装されているスキルの代表例を示す。

- 国会答弁案生成スキル：各部門において行われる国会答弁準備業務を支援する。国会議員からの質問に対して省庁は答弁を用意するが、過去の答弁との整合性が求められるため、担当者は過去の関連答弁を調査して内容を揃える必要があり、この作業に多大な工数がかかっていた。本スキルは DuckDB-Wasm を用いて過去の国会答弁データベースを自然言語で検索し、関連する過去答弁を引用根拠として明示しながら、整合性を保った答弁案を自動生成する。詳細は 5.5 節で述べる。
- 経路探索スキル：外部の物流ルート検索 API と連携し、トラック・船舶・鉄道を組み合わせたマルチモーダルな経路を探索・可視化する。地点名から座標への AI 解決、経路探索、DuckDB-Wasm への結果保存、地図への自動描画を一連のツール呼び出しで実行する。
- 回帰分析・クラスタ分析スキル：個別具体的な統計分析処理をスキルとして切り出し、ユーザーが自然言語で指示するだけで AI が適切な分析手法を選択・実行できるようにしている。

5.4　グラフ・地図可視化の実現

AI チャットで生成したデータ分析結果を直感的に理解するには、グラフや地図への可視化が不可欠である。しかし、AI が生成する可視化仕様にはいくつかの技術的課題がある。たとえば和暦形式のラベルが正しくソートされない問題、日本語カラム名の誤認識、行政データ特有のメッシュコード（JIS X 0410）への対応などである。本節では Vega-Lite・MapLibre GL JS・DuckDB-Wasm によるオンデマンド MVT 生成の 3 つの技術要素について、これらの課題に対する工夫を述べる。

5.4.1　Vega-Lite によるグラフ可視化

グラフ可視化ライブラリの選定については 5.2.3 項で詳述したとおり Vega-Lite を採用している。本節では AI チャット経由のグラフ生成を実現するための技術的な工夫点を述べる。

AI が Vega-Lite の仕様を生成する際の品質向上のため、スキルファイルにより詳細な実装パターンを提供する方式を採用している。一般的なグラフ生成の方法論はシステムプロンプトに含めず、「vega.basics」・「vega.sort」・「vega.gradient」等の個別スキルに記述する。AI はユーザーの要求に応じて必要なスキルを取得し、そのスキルに記述されたパターンに従って Vega-Lite 仕様を生成する。

特に日本の行政データに特有の課題として、和暦形式のソート問題への対応が必要であった。元号形式のラベル（例：「"H30.3"」、「"H30.11"」）をアルファベット順でソートすると「"H30.11" < "H30.3"」 という誤った順序になる。この問題に対応するため、「vega.sort-custom-order」 スキルには以下の SQL 生成パターンを定義した。

```
-- CREATE TABLE 実行時にソートキー列を付与する
SELECT *,
  CASE
    WHEN "月" LIKE 'H%.%' THEN
      (CAST(REGEXP_EXTRACT("月", 'H(¥d+)', 1) AS INTEGER) + 1988) * 100
      + CAST(REGEXP_EXTRACT("月", '¥.(¥d+)$', 1) AS INTEGER)
    WHEN "月" LIKE 'R%.%' THEN
      (CAST(REGEXP_EXTRACT("月", 'R(¥d+)', 1) AS INTEGER) + 2018) * 100
      + CAST(REGEXP_EXTRACT("月", '¥.(¥d+)$', 1) AS INTEGER)
    ELSE 999999
  END AS "_sort_order"
FROM "source_table"
```

この SQL 実行後、Vega-Lite のエンコーディングで 「"sort": {"field": "_sort_order", "op": "min", "order": "ascending"}」 を指定すると、元号形式のラベルを正しい時系列順に並べることができる。「"op": "min"」 を必須としているのはスタックグラフでのソート崩れを防ぐためであり、これを省略すると Vega-Lite がデフォルトで 「"op": "sum"」 を使用し、スタックセグメント数によってソートキーが変化して順序が崩れる問題が発生するためである。

5.4.2 MapLibre による地図可視化

地図可視化には MapLibre GL JS を採用している。本項では AI が地図の表示仕様を自動生成する際に直面した課題と対策を述べる。

AI が日本語カラム名を誤認識する問題

AI が MapLibre GL のスタイル式を生成する際、日本語のカラム名を「["get", "カラム名"]」の形式で参照する。しかし、LLM がテーブルのカラム名を正確に記憶・再現できないことに起因して、漢字の誤認識が発生する事例が確認された。1 文字の漢字の誤りでもスタイル式全体が機能しなくなるため、AI の生成精度に依存しない補正の仕組みが必要であった。
この問題に対して、スタイル式適用ツールにプロパティ名のファジーマッチング自動修正機能を実装した。AI が生成した式内の「["get", "fieldName"]」パターンを再帰的にスキャンし、「fieldName」が実際のカラム名と一致しない場合に最も類似度の高いカラム名へ自動修正する。この仕組みにより、AI の生成精度が完全でなくとも正しいスタイル式が適用される。

行政データ特有のメッシュコードへの対応

行政統計データでは、地域メッシュコード（JIS X 0410）を単位として集計されたデータが多数存在する。メッシュコードは緯度経度を符号化した整数値であり、そのままでは地図上にポイントや面として描画できない。AI にメッシュコードを含むデータの地図表示を指示した場合、AI が自律的にメッシュコードの地理的意味を解釈して適切な描画を行うことは期待できない。
この課題に対して、メッシュコードの自動解釈と地図描画の仕組みを実装した。メッシュコードを含むテーブルに対して地図生成を指示された場合、AI はデータ操作ツールにメッシュコードの使用列と次数（1 次〜4 次）を指定する。ツール内部でメッシュコードから対応する地理的な矩形領域を計算し、各メッシュをグリッド矩形として地図上に描画する。データ操作時に AI がメッシュコードのメタデータ（使用列と次数）を記録し、後続の地図スタイル更新やデータ参照でもメッシュ情報が引き継がれる設計としている。

地域メッシュコード（JIS X 0410）とは

地域メッシュコードとは、日本全国を階層的な格子状に分割する JIS 規格（JIS X 0410）で定められた地域識別コードである。コードは整数で表され、桁数に応じて 1 次（約 80km 四方）・2 次（約 10km 四方）・3 次（約 1km 四方）・4 次（約 500m 四方）の精度で地域を特定できる。国勢調査・交通量調査等の行政統計ではメッシュコードを単位として集計されたデータが多数存在するため、本システムではメッシュコードを地理的に解釈して地図上に可視化する機能を実装した。

データ加工時に地図スタイルが失われる問題

ユーザーが AI チャットで「このデータをフィルタして新しいテーブルを作って」のような指示を出した場合、元のデータに設定されていた地図スタイル（色分け・凡例等）が新しいテーブルに引き継がれず、ユーザーがスタイルを再設定しなければならないという問題があった。分析の過程ではデータの加工やフィルタは頻繁に行われるため、そのたびにスタイルが失われることは分析作業の流れを中断させ、操作の手間を増やす要因となる。
この問題に対して、マップスタイル自動継承の仕組みを実装した。AI がデータ加工を実行すると同時に、元テーブルの MapLibre スタイル定義（レイヤー・色設定・凡例等）が新テーブルに自動的にコピーされる。複数テーブルを結合した場合は、参照した全ソーステーブルのスタイルをレイヤー名でマッピングし統合する。スタイルで参照しているカラムが新テーブルに存在しない場合は、不足カラムを補完するための SQL を自動実行する。
これにより、データの加工・フィルタ操作を繰り返しても地図スタイルが維持され、ユーザーは分析作業を中断せずに継続できるようになった。

5.4.3　大規模地理空間データの描画性能

LINKS BI では多様な行政データを地図上に可視化する必要がある。数万件〜数十万件規模のポイントやポリゴンを含むデータも少なくないが、大規模な地理空間データを地図上に描画する際には性能上の課題がある。本項では従来方式の課題と、DuckDB-Wasm によるオンデマンド MVT 生成という解決策を述べる。

ベクタータイル（MVT）とは

地図上にデータを可視化する方式は大きく 2 つに分類される。1 つは GeoJSON 等のデータをそのままブラウザに読み込んで描画する方式であり、もう 1 つはベクタータイルと呼ばれる方式である。
ベクタータイルとは、地図の表示範囲とズームレベルに応じてデータを小さなタイル（区画）に分割し、必要な区画のみをブラウザに配信する仕組みである。MVT（Mapbox Vector Tile）はこの仕組みの標準フォーマットであり、Protocol Buffers 形式で効率的にエンコードされる。表示範囲外のデータは読み込まれないため、大規模データでも地図操作が軽快に保たれる。

従来方式の課題

従来の地図データ描画方式にはそれぞれ課題があった。
GeoJSON を直接描画する方式は実装が容易である一方、データ量が増大するとブラウザ側の描画処理が重くなり、地図の表示や操作が著しく遅くなるという問題がある。数万件〜数十万件規模のポイントやポリゴンを含む行政データでは実用に耐えない場合があった。
一方、従来のベクタータイル方式では、PostGIS や tippecanoe 等のツールを用いてデータを事前にタイル形式に変換し、タイルサーバーから配信するパイプラインを構築する必要がある。この方式はデータが追加・更新されるたびにタイルの再生成が必要であり、AI チャットのようにユーザーの質問に応じてリアルタイムにデータを加工・フィルタする利用形態には適さない。事前変換パイプラインと配信インフラの構築・運用にかかるコストも課題であった。
つまり、GeoJSON 直接描画では性能が不足し、事前タイル変換では柔軟性と即時性が犠牲になるというトレードオフが存在していた。

DuckDB-Wasm によるオンデマンド MVT 生成

このトレードオフを解消するため、LINKS BI ではブラウザ内の DuckDB-Wasm からベクタータイルをオンデマンドで生成し、MapLibre GL JS で表示するアプローチを採用した。DuckDB-Wasm v1.30.1 以降で利用可能になった「ST_AsMVT」および「ST_AsMVTGeom」関数により、DuckDB から MVT バイナリを直接生成できるようになったことで実現した手法である。事前にタイルデータを変換・配信する仕組みを一切持たず、ブラウザ内で SQL クエリの実行結果から MVT バイナリを直接出力する。

具体的には、MapLibre GL JS のカスタムプロトコル機能を利用し、独自スキーム（「duckdb://」）を登録する。MapLibre GL JS がタイルリクエストを発行すると、カスタムプロトコルハンドラがリクエストのズームレベル・タイル座標を解析し、対応する SQL クエリを DuckDB-Wasm に発行して MVT バイナリを取得・返却する仕組みである。

このアプローチにより前述のトレードオフが解消され、以下の利点が得られた。

- SQL クエリの変更が即座にタイル描画に反映されるため、探索的分析が高速になった。ユーザーが AI チャットでフィルタ条件を変更すると、次のタイルリクエスト時点で新しい条件に基づく MVT が生成される。事前タイル変換が不要であるため、データの加工・フィルタ操作とタイル描画が即座に連動する。
- サーバーサイドにタイル生成・配信パイプラインを持つ必要がなく、事前生成済みタイルの保存・管理も不要である。すべてのタイル生成がブラウザ内で完結するため、インフラ運用の負担が最小化された。
- SQL の柔軟性を活かして、任意のスキーマ変換・空間結合・集計をタイル生成と一体化できるため、分析の自由度が大幅に向上した。

描画性能の最適化

オンデマンド MVT 生成はブラウザ内で実行されるため、データ規模が大きくなるとタイル生成自体の処理時間が増加する。この問題に対して以下の最適化を実施した。

- ズームレベルに応じたジオメトリ簡略化を適用した。「ST_SimplifyPreserveTopology」関数を用い、低ズームレベル（広域表示）ではジオメトリの頂点数を削減して描画負荷を軽減し、高ズームレベル（詳細表示）では元のジオメトリを維持する。ズームレベル 15 以上では簡略化を無効にし、それ以下では段階的に簡略化の度合いを大きくする設計とした。
- R-Tree 空間インデックスの構築により、タイル境界内に含まれるフィーチャーの検索を高速化した。空間インデックスを使用しない場合、タイル生成のたびに全データのスキャンが必要となるが、インデックスにより必要なデータのみを効率的に抽出できる。
- タイルあたりのフィーチャー数に上限（50,000 件）を設け、極端に密集した領域でもブラウザの描画処理が破綻しないようにした。

一方、DuckDB-Wasm のタイル生成はブラウザのメモリに依存するため、数百万ポリゴン規模の超大規模データセットに対しては、Parquet 形式による列指向アクセスや事前集約による行数削減を組み合わせることが推奨される。データ規模とユースケースに応じて、オンデマンド MVT 生成と事前タイル変換を使い分ける運用が現実的である。

5.5　国会答弁案生成の性能最適化

5.5.1　最適化の背景

国会答弁案生成機能は、LINKS BI の AI チャットにおける最も複雑なスキルの 1 つである。国会答弁案モードに入ると、過去の国会答弁書データ（質問者・答弁者・質問本文・答弁本文等を含むテーブル）が BI に自動的に読み込まれる。ユーザーが国会質問の趣旨を入力すると、AI がこの答弁書テーブルを参照しながら以下の 3 段階の処理を順に実行して答弁案を自動生成する。

1. 素材の収集：答弁書テーブルをテキスト検索し、質問の趣旨に関連する過去答弁を抽出する。抽出結果は素材テーブルとして保存される。
2. 構成の生成：収集した素材をもとに、答弁案の段落構成（各段落の役割・トピック・引用する過去答弁）を決定する。構成は構成テーブルとして保存され、各段落に質問者 ID・答弁者 ID・引用元答弁全文などのメタデータが紐づく。
3. 答弁案の出力：構成テーブルに基づいて最終的な答弁案テーブルを生成し、バリデーション（文字数・段落の役割・カラム構成の検証）を経てユーザーに表示する。

この処理は複数のツールを連続的に呼び出すマルチステップの処理であり、初期実装時点では 1 回の答弁案生成に 104 秒を要していた。行政業務における実用性を確保するためには、大幅な所要時間短縮が必要であった。

5.5.2　初期実装の課題と最適化施策

AI の処理時間は「読み込む情報量」「生成するテキスト量」「システムとの往復回数」の 3 要素で構成される。初期実装を分析した結果、これらの要素それぞれに改善余地のある課題が特定された。以下に主要な課題と施策を示す。

AI の出力量が過大であった問題

AI が構成テーブルを作成するたびに 100 行超の SQL 文を毎回生成しており、質問者・答弁者の識別情報や引用元答弁全文といった定型データが段落ごとに繰り返されていた。また、素材収集と構成生成が別々のツール呼び出しで実行されており、ツール間の AI 思考時間も余分に発生していた。

- 対策として、答弁案構成に特化した専用ツールを設計し、AI は JSON 形式で最小限の情報のみを渡す方式に変更した。定型データはツール内部で自動取得する。さらに素材収集と構成生成を 1 回のツール呼び出しに統合した。
- この変更により、構成テーブル生成の所要時間が 89 秒から 35 秒に短縮された。

不要なツール呼び出しが多数発生していた問題

固定であるテーブル構造の確認（3 回、約 8 秒）やガイドラインの取得（1 回、約 2 秒）が毎回発生していた。また、テキスト検索ツールが複合条件に非対応であったため、AI が論点ごとに個別の検索を実行して複数回の往復が発生していた。

- 対策として、テーブル構造を AI の指示文に事前記載し、ガイドラインを初期メッセージに埋め込んで不要な取得を省略した。テキスト検索ツールには OR 条件での一括検索機能を追加し、検索回数を 3 回から 1 回に削減した。
- これらの変更により、合計約 13 秒分の不要な往復が解消された。

AI の行動が不安定であった問題

専用ツールを導入しても、汎用ツールで SQL 文を直接記述したり、ツールを一切使わず答弁案を直接出力したりする不安定な挙動が約 30%の確率で発生していた。前者は汎用ツールの説明文が圧倒的に長く AI の注意を分散していたこと、後者は出力形式の定義とツール使用手順を同じ場所に配置したことで AI が「ツール不要」と判断したことが原因であった。

- 対策として、専用ツールの説明文を充実させ、モード固有の指示が汎用規則を上書きできる例外規定を設けた。さらに AI への指示を「出力形式の定義」と「ツール使用手順」の 2 層に分離し、出力形式を知っていてもツール使用を省略できない構造とした。
- これによりツール未使用率が 30%から 0%に改善し、所要時間が 71 秒から 57 秒に短縮された。

出力品質の制御が不安定であった問題

LLM はトークン単位で生成するため文字数を正確に守れず、上限超過によるバリデーションエラーと再試行（1 回あたり 60〜90 秒の遅延）が頻発していた。また、テキスト検索が全カラム・全行を返す仕様のため、検索結果が約 86,000 文字に達して AI の処理能力を圧迫していた。

- 文字数制御では、目標値と上限値の両方を AI に提示する方式を採用した。さらに AI に見せる上限値（650 字）とシステム内部の拒否基準（700 字）を分離してバッファを設けた。
- テキスト検索では、返却カラムの限定・長文値の截断（200 字）・レスポンスサイズの上限設定を実施し、検索結果を 11,000 文字に削減した（87%削減）。

これにより。再試行が解消され、最悪ケースの所要時間が 230 秒から 74 秒に改善した。

5.5.4　最適化の推移と成果

以下に最適化の取り組みフェーズごとの答弁案生成にかかった所要時間の推移を示す。

表 5.2　最適化フェーズごとの所要時間の推移

フェーズ	所要時間	主な施策
ベースライン	104 秒	初期実装
フェーズ 1	254 秒	専用ツール導入＋text_search OR 対応（一時的な悪化）
フェーズ 2	184 秒	RULE ZERO 例外＋description 充実
フェーズ 3	97 秒	テーブル探索省略＋ガイドライン初期メッセージ化
フェーズ 4	77 秒	ツール統合（素材＋構成）＋SELECT 省略
フェーズ 5	71 秒	文字数安定化＋テキスト検索サイズ制限

最終	57 秒	知識と手段の分離

フェーズ 1 にて一時的な性能悪化は見られたものの、続く継続的な改善により処理速度は大幅に向上した。

以下にベースラインと最終状態の処理ステップ別比較を示す。（AI が実際に答弁案生成をするための一連の流れである。）

表 5.3　処理ステップ別のベースラインと最適化後の比較

処理ステップ	ベースライン	最適化後	改善
ガイドライン取得	2 秒 （ツール呼び出し）	0 秒 （初期メッセージ 埋め込み）	-2 秒
テーブル確認 （duckdb_query × 3）	8 秒	0 秒 （システムプロンプトに 既知情報記載）	-8 秒
素材テーブル作成	23 秒（SQL 生成）	統合	—
構成テーブル作成	37 秒（SQL 生成）	28 秒（JSON 出力）	-9 秒
バリデーション＋調整	10 秒	3 秒	-7 秒
最終 SELECT	5 秒	0 秒	-5 秒
最終メッセージ	18 秒	16 秒	-2 秒
合計	**104 秒**	**57 秒**	**-45%**

最終的には初期実装比で 45%の所要時間短縮を達成した。

5.6　UI/UX の改善

5.6.1　2024 年度の課題

2024 年度の LINKS BI では、汎用的なデータ可視化基盤が存在せず、ユースケースごとに個別のフロントエンドアプリケーションを開発する運用をとっていた。各アプリは特定の業務要件に特化して構築されたものであり、新たなユースケースが生じるたびに開発コストが発生するという構造的な問題を抱えていた。この運用形態の結果として、以下の 2 点が主要な課題として顕在化した。

- 個別アプリの操作性の低さ：ユースケース別に独立して開発されたアプリでは、集計対象のカラム指定・フィルター条件の設定・グラフ種別の選択といった操作をユーザーが自ら判断・入力する必要があった。これらの操作はデータ構造やシステムの仕様に関する一定の理解を前提とするものであり、技術的な専門家でない国交省職員にとって学習コストが

高く、利用を途中で断念するケースが多かった。また、ユースケースをまたいで UI や操作フローが統一されていないため、アプリを切り替えるたびに操作を学び直す必要があり、習熟の妨げにもなっていた。

- 大規模データの処理性能：100 万行規模のデータの読み込みや集計処理に時間がかかり、ユーザーが分析の流れを中断せざるを得ない状況が発生し、業務での継続利用を妨げる要因となっていた。

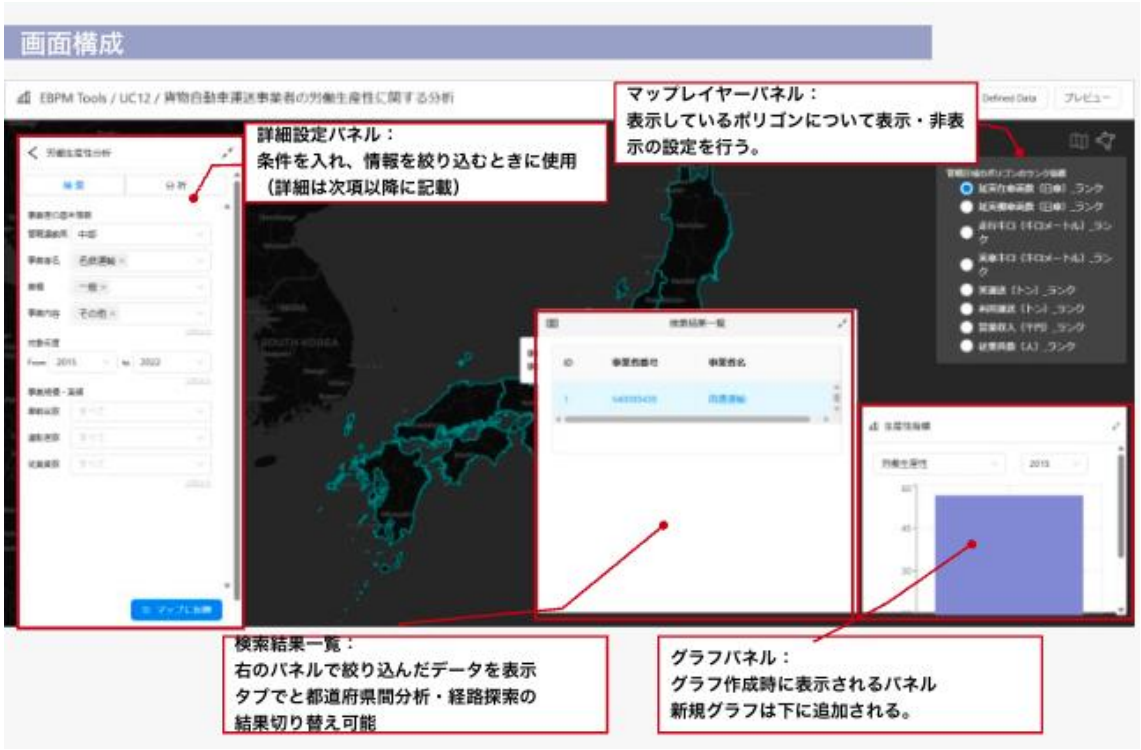


図 42 2024 年度の BI 画面

5.6.2　2025 年度の実施改善

2025 年度では LINKS BI もゼロから再設計・再実装し、上記の課題に対して以下の改善を実施した。

- データ活用モジュール（LINKS BI）への一本化と AI チャットの導入：ユースケース別個別アプリを廃止し、すべての業務に対応できる汎用のデータ活用モジュール（LINKS BI）へ一本化した。操作性改善の中核となる機能が、AI チャットによる自然言語での可視化操作である。従来はユーザーがカラム名・集計軸・グラフ種別を自ら選択する必要があったが、新しい設計ではユーザーがチャット欄に業務上の問いをそのまま入力するだけで、AI が適切なクエリを自動生成し、グラフや地図として出力する。例えば「都道府県別の事故件数を多い順に棒グラフで見せて」「過去 1 年間の月別推移を折れ線で表示して」といった日常的な言葉で指示できるため、データ構造や SQL の知識を持たない職員でも分析操作が完結する。カラム名の正確な入力や集計方法の選択といった技術的ハードルが排除されたことで、断念ケースの大幅な削減が期待される。

図 43 2025 年度の BI 画面（チャット画面）

図 44 2025 年度の BI 画面（ダッシュボード画面）

- DuckDB-Wasm によるブラウザ内データ処理：DuckDB-Wasm を採用し、ブラウザ内で SQL を実行する設計に刷新した。従来の設計ではデータの読み込みや集計のたびにサーバーへのリクエストが発生し、ネットワーク遅延やサーバー負荷が応答時間に影響していた。新設計ではデータをブラウザにロードした後の集計・フィルタリング操作はすべてローカルで完結するため、操作に対する応答が大幅に改善された。（詳細は 5.1 節参照）。

これらの改善がユーザー体験にどう寄与したかの評価は、第 6 章 6.3 節で述べる。

5.7　AI チャット機能の精度および性能検証

5.7.1　精度および性能検証の背景と目的

本節では、AI チャット機能について以下の 2 点を検証した結果を記録する。

- AI チャットとしての精度検証：ユーザーの自然言語入力に対して AI が正しいツールを正しい引数で呼び出せるかを定量的に評価する。Function calling によるツール実行の正確性を回帰テストにより継続的に計測し、品質水準を数値で示す。
- BI アプリケーションとしての性能検証：データ容量・連続処理量・同時接続数における安定稼働の限界を確認する。処理性能の検証結果は 5.7.3 項で述べる。

検証は以下の 6 つのデータを対象に実施した。

- ①貨物自動車運送事業実績報告データ
  - 2024 年問題を背景に労働生産性の向上が求められる中、事業者別の営業実績・人件費・財務情報の集計・分析効率化を目的に、管理貨物自動車運送事業者が提出する事業実績報告書および事業報告書を構造化したもの。
- ②貨物流動・交通ネットワークデータ
  - 物流分野のカーボンニュートラル推進に向けたモーダルシフト分析を目的に整備された、品目別・交通モード別の貨物流動データおよび鉄道・航路ネットワークのジオメトリデータ。経路探索機能のデータとして使用される。
- ③無人航空機飛行計画データ
  - 無人航空機の安全施策検討を目的に収集された、飛行計画申請データおよび事故等報告データ。
- ④内航海運事業報告・船員データ
  - 船員確保や待遇改善に向けた施策立案を目的に、事業状況報告および内航海運業事業概要報告書を構造化したもの。
- ⑤国会答弁書データ
  - 答弁案作成業務の効率化・質向上を目的に、各担当部局が作成する Word 形式の答弁書を構造化したもの。
- ⑥鉄軌道事業報告・事故報告データ
  - 鉄軌道事業の実態把握と事故防止施策の検討を目的に、事業者が提出する事業概況報告書・実績報告書および運転事故等報告書を構造化したもの。

5.7.2　AI チャットの精度検証

本項では、AI チャット機能のツール呼び出し精度に関する検証の設計・手法・結果を記述する。

検証の設計と観点

LLM の出力には本質的なランダム性がある。同一の入力に対して常に同一の出力が得られるわけではなく、指示の解釈や処理手順の選択に揺らぎが生じる。一方で、本プロジェクトの実証を通じて実際のユーザーが運用に活用することを考えると、使用するたびに結果が変わるシステムは行政の業務利用に耐えない。再現性のある形で結果が得られることが実用化の前提条件であるため、この揺らぎを「失敗」として単純に扱うのではなく、「N 回実行して何%成功したか」という統計的な指標を採用し、一定確率以上で正しい結果を生成することを合格条件とした。このランダム性に対しては、5.2.4 項で述べた temperature=0 の設定とスキルシステム

による手順の固定化で対処している。これらの施策により、99%再現率の達成を現実的な目標とした。

評価対象ツールと評価方式

本節で評価対象とする「ツール」とは、5.2.1 項で解説した Function calling（関数呼び出し）の仕組みにおいて AI が呼び出せる関数のことを指す。AI はユーザーの自然言語による指示を解釈し、適切なツールを選択してその「インプット（引数）」すなわちツールに渡すパラメータ（SQL 文、グラフ仕様、分析条件等）を生成する。本検証ではこのツール選択の正確性とインプット生成の妥当性を評価する。

評価対象の全ツールと評価方式の対応を以下に示す。

表 5.4　評価対象ツールと評価方式

No.	ツール名	カテゴリ	評価方式	ツールの概要
1	「duckdb_query」	データ処理	出力検証（出力テーブル検証）	DuckDB を利用した SQL の実行
2	「duckdb_text_search」	データ処理	入力検証	DuckDB を利用したテキスト検索
3	「update_vega_graph_spec_for_table」	可視化	入力検証（可視化タイプ・軸）	Veda-Lite を利用したグラフ可視化
4	「update_map_style_for_table」	地図	入力検証（ジオメトリ・スタイル）	Maplibre GL JS を利用したスタイル付き地図可視化
5	「perform_cluster_analysis」	統計分析	入力検証（引数・パラメータ）	クラスター分析の実行
6	「select_predictors_for_regression」	統計分析	入力検証（引数・パラメータ）	回帰分析における説明変数の推定
7	「perform_regression_analysis」	統計分析	入力検証（引数・パラメータ）	回帰分析の実行
8	国会答弁案生成（処理フロー全体）	文書生成	総合検証（最終出力の構造・内容）	国会答弁案の生成

No.1～7 は特定のツール単体の動作を検証するものである。一方、No.8 の国会答弁案生成は特定のツールではなく、5.5.1 節で述べた 3 段階の処理フロー全体を通して複数のツール（テキス

ト検索・構成テーブル作成・バリデーション等）を連鎖的に呼び出した末の最終出力を検証するものであり、他の項目とは性質が異なる。

評価方式の定義は以下のとおりである。

- 出力検証（「duckdb_query」 対象）：生成テーブルの構造（カラム構成・行数）および統計値（合計・最大・最小）が事前に定めた基準値と一致するかを検証する。SQL 文の内容・構文は評価対象外とする。同一の処理意図に対して等価な複数の SQL 表現が存在するためである。」
- 入力検証（その他ツール対象）：ツールへ渡される引数（Function calling の Input オブジェクト）に、必要なプロパティが正しい値・形式で含まれているかを検証する。処理内部の実装詳細は問わない。
- 総合検証（国会答弁案生成）：処理フロー全体を通した最終出力（答弁案テーブル）の構造（カラム数・段落の役割）と内容（文字数・キーワードの包含）が基準を満たすかを検証する。個別のツール呼び出しの正否ではなく、一連の処理フロー全体の成否を評価する。

その他、共通の「成功」判定基準は以下のとおりである。

1. 必須ツールが呼び出されている（他のツールへの代替呼び出しが発生していない）。
2. ツールの引数に必須プロパティが存在し、値の内容が期待どおりである。
3. 出力検証の場合、出力テーブルの行数・カラム構成・統計値が基準値と一致している。

ツール別検証方法

各ツールの検証方法を以下に示す。

duckdb_query（SQL クエリ）

AI が生成する SQL 文は同一の処理意図に対して複数の等価な表現が存在するため、SQL 文の内容・構文は評価対象としない。代わりに出力検証として、ツール実行後に生成されるテーブルの構造と統計値が基準値と一致するかを検証する。たとえば事故件数を含むテーブルに対して「都道府県別に事故件数を集計して」と指示した場合、AI は SQL 文を生成して集計テーブルを出力する。検証ではこの出力テーブルに対して、事故件数の合計値が事前に計算した正解値（例：全国合計 1,234 件）と一致するか、都道府県別の最大値・最小値が期待どおりか、行数が 47 都道府県分であるかを確認する。AI が内部で生成する SQL 文は「GROUP BY prefecture」でも「GROUP BY 1」でも構わない。出力テーブルの数値が毎回同一であれば、内部の SQL 表現が多少異なっても意味的に等価なクエリを発行していると判断できる。逆に、集計値が実行ごとに異なる場合は、AI が異なるフィルタ条件や集計方法を選択している（揺らぎが発生している）ことを意味する。

主な確認項目：ツール呼び出しの有無、生成テーブルの行数・カラム構成、集計値（合計・最大・最小）の一致、実行時間の計測。

update_vega_graph_spec_for_table（グラフ可視化）

Vega-Lite のレンダリング結果は実装・設定値によって多様な表現が存在するため、グラフ外観（色・フォント等）は評価対象としない。AI がツールに渡す Vega-Lite スペックのうち「可

視化タイプの選択（mark）」「X 軸・Y 軸・色分け軸のフィールド指定（encoding）」「集計方法（aggregate）」が指示内容に合致しているかを入力検証する。なお本ツールは「get_skill」による「vega」ドメインの取得が前提条件であるため、そのステップも確認対象に含める。

update_map_style_for_table（地図可視化）

MapLibre のスタイル定義は多様な表現方法が存在するため、スタイルの外観（色の具体値・線幅等）は評価対象としない。AI が生成する MapLibre スタイルオブジェクトに「ジオメトリ種別（point/line/polygon）」「レイヤータイプ」「スタイルプロパティのカラム参照」が適切に含まれているかを入力検証する。

perform_cluster_analysis（クラスタリング分析）

K-means クラスタリングのアルゴリズム動作は評価対象外。AI がツールに渡す「クラスタリング対象カラム（feature_columns）」「クラスタ数（k）」「対象テーブル名」が指示と合致しているかを入力検証する。

select_predictors_for_regression / perform_regression_analysis（回帰分析）

回帰分析のアルゴリズム動作は評価対象外。AI が事前ステップとして「select_predictors_for_regression」で適切な説明変数候補を選定し、続く「perform_regression_analysis」に「目的変数」「説明変数」が正しく渡されているかを入力検証する。

検証シナリオとカバレッジ

各業務から 2〜3 シナリオをピックアップし、ツール種別・操作複雑度・可視化タイプの組み合わせが偏りなくカバーされるよう選定した。全 20 シナリオの一覧を以下に示す。

表 5.5　検証シナリオ一覧（全 20 シナリオ）

No.	対象データ	テスト名	ツール	操作複雑度
S1-1	鉄軌道事業報告・事故報告	自然災害事故の絞り込み	「duckdb_query」	1 テーブル×複合操作
S1-2	鉄軌道事業報告・事故報告	自然災害事故路線のジオメトリ結合	「duckdb_query」	複数テーブル×結合
S2-1	鉄軌道事業報告・事故報告	強風キーワード検索	「duckdb_text_search」	テキスト検索
S3-1	鉄軌道事業報告・事故報告	路線別被害額棒グラフ（色分け NULL 補完）	「update_vega_graph_spec_for_table」	棒グラフ（単一）

S3-2	国会答弁書	答弁件数の年月別積み上げ棒グラフ	「update_vega_graph_spec_for_table」	積み上げグラフ（色分け＋集計）
S3-3	内航海運事業報告・船員	時間単価×労働時間の 2 軸折れ線グラフ	「update_vega_graph_spec_for_table」	折れ線グラフ（時系列）
S4-1	無人航空機飛行計画	飛行計画の出発地ポリゴン地図	「update_map_style_for_table」	ポイント
S4-2	鉄軌道事業報告・事故報告	自然災害路線のライン地図（運休本数色濃淡）	「update_map_style_for_table」	ライン（ジオメトリ＋色分け）
S5-1	貨物自動車運送事業実績報告	2 変数クラスタリング（結合済データ）	「perform_cluster_analysis」	単一指示
S6-1	貨物自動車運送事業実績報告	実車キロ当たり事故件数の回帰分析（単純）	「select_predictors」 / 「perform_regression」	単一指示
S7-1	国会答弁書	気候変動防災の国会答弁案作成	国会答弁案生成（処理フロー全体）	単一指示
S8-1	貨物自動車運送事業実績報告	近畿×営業収入 MAX 抽出	「duckdb_query」	複合集計
S8-2	鉄軌道事業報告・事故報告	大雨土砂 AND 検索＋ソート	「duckdb_query」	フィルタ＋ソート
S9-1	貨物自動車運送事業実績報告	説明変数指定＋除外カラム付き回帰分析	「update_vega_graph_spec_for_table」	散布図
S9-2	鉄軌道事業報告・事故報告	JR 各社営業収益の棒グラフ	「update_vega_graph_spec_for_table」	ヒートマップ
S10-1	無人航空機飛行計画	目的地ポリゴン地図（10 段階色階級）	「update_map_style_for_table」	ポリゴン＋色分け

S11-1	鉄軌道事業報告・事故報告	台風×2 テーブル突き合わせ検索	「duckdb_query」	和暦ソート
S12-1	無人航空機飛行計画	無人航空機事故のポリゴン集計地図	「duckdb_text_search」	複合条件
S13-1	貨物流動・交通ネットワーク	神戸港→北九州港の経路探索	経路探索ツール	単一指示
S14-1	貨物自動車運送事業実績報告	結合済データで事故件数回帰分析	「duckdb_query」	メッシュコード変換

カバレッジの観点：全 8 ツールに最低 1 シナリオ、単純・複合・結合の操作パターン、棒グラフ・折れ線・積み上げ・散布図・ヒートマップの複数 Vega-Lite タイプ、ポイント・ライン・ポリゴンの複数ジオメトリタイプ、「get_skill」経由ツールが各 1 シナリオ以上。

各シナリオは 10 回ずつ実行し、再現率（成功回数/ 10 回 × 100%）で評価する。合格基準は全シナリオで 99%以上（10 回中 10 回成功）とした。試行回数を n=10 とした根拠は次のとおりである。LLM の出力は確率的であるため、統計的に厳密な 99%再現率の保証には本来より大きなサンプルサイズが必要となる（例：二項分布において 95%信頼区間で再現率 99%以上を保証するには n≧300 程度が必要）。しかし、AI チャットの 1 回の検証実行には LLM API の呼び出しとブラウザ内でのツール実行を伴うため、大量のサンプルを確保するには相応のコストと時間を要する。本検証では n=10 を採用し、全 10 回の成功をもって「実用上十分な再現性がある」と判断する基準とした。これに加えて、開発中の回帰テスト（n=5・成功率 80%以上）による継続的な品質監視を併用することで、2 段階の品質管理体制を構築している。

検証の実施方法（AI 回帰テストフレームワーク）

上述の検証は、独自に構築した AI 回帰テストフレームワーク上で実施した。本フレームワークは Vitest +ブラウザ環境（Playwright）で構成されており、ブラウザ内で DuckDB-Wasm を初期化し、実際のデータファイルをインポートした上で AI チャットを動作させることで、本番環境と同一の条件での検証を実現している。前述の「検証シナリオとカバレッジ」で示した 20 シナリオは、このフレームワーク上にテストケースとして実装されている。

なお、本フレームワークは本検証専用ではなく、開発時の継続的な品質保証にも活用している。開発中は合格基準を緩和し（5 回実行・成功率 80%以上）、プロンプトやスキルの変更が既存の動作に影響しないかを素早く確認する回帰テストとして運用している。

1. DuckDB-Wasm の初期化
2. テスト用 CSV データをテーブルにインポート
3. ツール群の初期化（本番環境と同一のツールを登録）
4. システムプロンプトの生成
5. N 回ループ実行
    1. ツール群の状態をリセット

2. AI との会話を実行
3. AI のツール呼び出しをブラウザ内で実行（DuckDB-Wasm 含む）
4. verify 関数で結果を検証（グラフ生成有無・データ内容・ソート順等）
5. 合否を記録

6. 成功率計算・レポート出力（Markdown・JSON）

上記フローのうち、合否判定の中核を担うのが verify 関数である。verify 関数はテストケースごとに定義され、AI の実行結果を受け取って検証項目の合否を返す。すべての検証項目が合格であればそのテストケースは成功、1 つでも不合格があれば失敗と判定する。

たとえば棒グラフの生成テストでは、verify 関数が「グラフが 1 つ以上存在すること」「Vega-Lite の mark 属性が「bar」であること」（棒グラフを可視化する入力となっていること）を検証する。SQL クエリのテストでは、verify 関数が DuckDB-Wasm に検証用 SQL を実行し、結果テーブルの行数・カラム構成・統計値を基準値と照合する。このように verify 関数はテストケースの性質に応じて柔軟に検証ロジックを記述できる設計となっている。

検証結果

全 20 ケースを各 10 回実行した結果、最終的に全ケースで 100%の成功率を達成し、全シナリオが合格基準（99%以上）を満たした。

ただし、初回の検証実行時点では全 20 ケース中 7 ケースが 100%に到達していなかった。たとえば S2-1 の強風キーワード検索は成功率 40%、 S3-3 の 2 軸折れ線グラフは 60%にとどまっていた。これらのケースに対して「スキルチューニング」を実施し、最終的に全ケースで 100%を達成した。スキルチューニングとは、各ケースの検証結果を分析して失敗原因を特定し、該当するスキルファイル（AI への指示文）に MUST/NEVER ルールや具体的な手順制約を追記することで、AI の動作を安定化させる改善プロセスである。以下の表では、スキルチューニングを実施したケースを※マークで示す。

表 5.6　検証結果（全 20 ケース×10 回実行）

No.	対象データ	テスト名	ツール	再現率	備考
S1-1	鉄軌道事業報告・事故報告	自然災害事故の絞り込み	「duckdb_query」	100%	—
S1-2	鉄軌道事業報告・事故報告	自然災害事故路線のジオメトリ結合	「duckdb_query」	100%※	duckdb/join.md にキープレビュー・完全性検証ルール追加
S2-1	鉄軌道事業報告・事故報告	強風キーワード検索	「duckdb_text_search」	100%※	analysis.md に_id ベース再構築ルール＋text-search.ts 修正

S3-1	鉄軌道事業報告・事故報告	路線別被害額棒グラフ（色分け NULL 補完）	「update_vega_graph_spec_for_table」	100%	—
S3-2	国会答弁書	答弁件数の年月別積み上げ棒グラフ	「update_vega_graph_spec_for_table」	100%	—
S3-3	内航海運事業報告・船員	時間単価×労働時間の 2 軸折れ線グラフ	「update_vega_graph_spec_for_table」	100% ※	vega.basics に SQL 事前集計 MUST ルール追加
S4-1	無人航空機飛行計画	飛行計画の出発地ポリゴン地図	「update_map_style_for_table」	100%	—
S4-2	鉄軌道事業報告・事故報告	自然災害路線のライン地図（運休本数色濃淡）	「update_map_style_for_table」	100%	—
S5-1	貨物自動車運送事業実績報告	2 変数クラスタリング（結合済データ）	「perform_cluster_analysis」	100%	k-means 想定内
S6-1	貨物自動車運送事業実績報告	実車キロ当たり事故件数の回帰分析（単純）	「select_predictors」 / 「perform_regression」	100% ※	regression/workflow.md に JSON 配列処理ルール追加
S7-1	国会答弁書	気候変動防災の国会答弁案作成	国会答弁案生成（処理フロー全体）	100% ※	validate_tobenan に SCHEMA/ROLES チェック追加
S8-1	貨物自動車運送事業実績報告	近畿×営業収入 MAX 抽出	「duckdb_query」	100%	—
S8-2	鉄軌道事業報告・事故報告	大雨土砂 AND 検索＋ソート	「duckdb_query」	100%	—
S9-1	貨物自動車運送事業実績報告	説明変数指定＋除外カラム付き回帰分析	「update_vega_graph_spec_for_table」	100%	—

S9-2	鉄軌道事業報告・事故報告	JR 各社営業収益の棒グラフ	「update_vega_graph_spec_for_table」	100% ※	data-guidelines に重複排除・単位保持ルール追加
S10-1	無人航空機飛行計画	目的地ポリゴン地図（10 段階色階級）	「update_map_style_for_table」	100%	—
S11-1	鉄軌道事業報告・事故報告	台風×2 テーブル突き合わせ検索	「duckdb_query」	100%	—
S12-1	無人航空機飛行計画	無人航空機事故のポリゴン集計地図	「duckdb_text_search」	100%	—
S13-1	貨物流動・交通ネットワーク	神戸港→北九州港の経路探索	経路探索ツール	100%	—
S14-1	貨物自動車運送事業実績報告	結合済データで事故件数回帰分析	「duckdb_query」	100% ※	regression/workflow.md に JSON 配列→1 行/エンティティルール追加

数値指標の安定性分析

verify 関数は合否判定に加えて、テーブルの行数・集計値・文字数などの数値を記録することもできる。これらの数値は合否には影響しないが、実行ごとのばらつきを把握することで AI の出力の安定性を評価できる。

10 回実行における数値指標の安定性を分析した結果、大半の指標が全 10 回で同一値を示した。以下に主要な安定性データを示す。

表 5.7　全回同一値を示した指標（抜粋）

テスト	指標	値
貨物自動車運送事業実績報告データ：クラスタリング	メインテーブル行数	39,139
貨物自動車運送事業実績報告データ：単純回帰	メインテーブル行数	39,826
貨物自動車運送事業実績報告データ：結合後回帰	メインテーブル行数	40,262

無人航空機飛行計画データ目的地ポリゴン	行数	1,902
無人航空機飛行計画データ：ドローン事故	集計対象市区町村数/事故件数最大値	620 / 8
無人航空機飛行計画データ：飛行計画	集計対象市区町村数/飛行計画総数	1,886 / 1,140,776
国会答弁書データ：積み上げ棒	行数/答弁件数合計	564 / 5,137
鉄軌道事業報告・事故報告データ：ジオメトリ結合	行数/ユニーク路線名数	234 / 3

表 48　ばらつきが確認された指標

テスト	指標	範囲	中央値	原因
貨物自動車運送事業実績報告データ：クラスタリング	最大クラスタ比率	97.69〜99.31%	98.07%	k-means の初期値ランダム性（想定内）
国会答弁書データ：国会答弁	段落数	4〜5	4	LLM の生成揺らぎ（全回 合格）
国会答弁書データ：国会答弁	答弁案文字数	547〜696 字	597 字	LLM の生成揺らぎ（全回 合格）

ばらつきが確認された項目はいずれも検証の合格基準内に収まっており、テスト結果には影響していない。k-means クラスタリングのばらつきはアルゴリズムの初期値ランダム性に起因する本質的な性質であり、AI の問題ではない。国会答弁案の文字数・段落数のばらつきは LLM の生成における固有の揺らぎであるが、バリデーションツールによる自動チェックにより全回が合格判定を得ている。

スキルチューニング実績

以下にスキルチューニングで解決した主要な問題と施策を示す。

表 5.8　スキルチューニング実績一覧

対象	問題	施策	効果
内航海運事業報告・船員データ：2 軸折れ線	AI が Vega-Lite transform.aggregate で直接集計	vega.basics に SQL 事前集計 MUST ルール追加	60%→100%

国会答弁書データ：国会答弁	カラム数が 9〜19 でバラバラ	validate_tobenan に SCHEMA/ROLES チェック追加	80%→100%
鉄軌道事業報告・事故報告データ：ジオメトリ結合	JOIN 名前不一致で行数ブレ	duckdb.join にキープレビュー・完全性検証ルール追加	不安定→100%
鉄軌道事業報告・事故報告データ：JR 営業収益	年度フィルタ・重複・単位変換のブレ	data-guidelines に重複排除・単位保持ルール追加	60%→100%
鉄軌道事業報告・事故報告データ：強風検索	text_search 後の SQL 追加フィルタで検索漏れ	analysis.md に_id ベース再構築ルール＋ text-search.ts 修正	40%→100%
貨物自動車運送事業実績報告データ：回帰分析	JSON 配列展開方法のブレで行数変動	regression.workflow に JSON 配列→1 行/エンティティ MUST ルール追加	不安定→100%

以下に代表的なチューニング事例の問題発見から解決までのプロセスを記述する。

事例 1： S3-3 **内航海運事業報告・船員データ　2 軸折れ線グラフ（60%→100%）**

AI が 40%の確率で Vega-Lite の 「transform.aggregate」 を使ってグラフ定義内で直接集計し、SQL による集計テーブルを作成しなかった。この場合、集計結果がグラフ仕様内に埋め込まれるため、verify 関数が集計テーブルの行数・カラム構成を検証できず不合格となった。「vega/basics.md」 スキルに「データ集計は MUST で SQL で行うこと、Vega-Lite transform.aggregate を使わないこと」ルールを追加した結果、全 10 回で SQL 集計テーブル（12 行×3 列）が作成され、数値も全回完全一致（時給 2,406.36〜3,179.30、労働時間 188.98〜213.36）に安定した。

事例 2： S7-1 **国会答弁書データ 国会答弁案作成（80%→100%）**

本事例のチューニングは 5.5 節で述べた国会答弁案生成の抜本的な性能最適化（104 秒→57 秒）の実施前に行ったものであり、当時のツール構成を前提としている。答弁案テーブルのカラム数が実行ごとに 9〜19 でバラバラになり、段落の役割（第 1 段落＝現状認識、最終段落＝今後の方針）が不正になる問題が発生していた。「validate_tobenan」 ツールに SCHEMA verdict（カラム数が 14 未満の場合に再作成を指示）と ROLES verdict（段落の役割が不正の場合に修正を指示）のチェックを追加した。さらにガイドラインに「全 14 カラム必須」「新規作成の段落でも NULL 指定で 14 カラムを含める」を明記した。チューニング後、カラム数は全 10 回で 14 に安定し、段落の役割も 100%正しく設定された。

事例 3： S2-1 **鉄軌道事業報告・事故報告データ　強風キーワード検索（40%→100%）**

2 段階の修正で安定化した。第 1 段階として、「workflow/analysis.md」 スキルに「キーワード検索時は MUST で 「duckdb_text_search」 ツールを使用し、自前の SQL LIKE/ILIKE で代替しないこと」ルールを追加した。しかしそれだけでは不十分であった。AI が「duckdb_text_search」 の結果からタイムスタンプ値で再フィルタする際に BIGINT→TIMESTAMP 型変換エラーが発生していたためである。第 2 段階として、「duckdb_text_search」 ツール自体を修正し、「returnColumns」 指定時にも常に 「_id」 カラムを返すようにした。スキルには「_id」 を使って 「WHERE _id IN (...)」 でテーブル再構築すること」ルールを追加した。型不一致エラーのリスクなく確実にテーブルを再構築できるようになり、全 10 回で 6 行を安定取得した。

事例 4： S9-2 **鉄軌道事業報告・事故報告データ　JR 各社営業収益（60%→100%）**

3 つの問題が同時に発生していた。(1)年度フィルタで 「LIKE '%2024%'」 が使われ 7 行になるケース、(2) JR 北海道が 2 行に重複するケース、(3)営業収益が百万円単位に変換され値が不一致になるケースである。「duckdb/date-handling.md」 に「年度テキストの完全一致ルール（「= '2024 年度'」）」、「workflow/data-guidelines.md」 に「ROW_NUMBER()による重複排除ルール」と「元の数値を保持し単位変換禁止ルール」を追加した。さらに、「vega/basics.md」 と 「duckdb/date-handling.md」 に 「deps: workflow.data-guidelines」 を追加し、グラフ作成・日付処理時に自動的にデータガイドラインが届くようにした。チューニング後、全 10 回で行数 6・全社収益値が完全一致した。

以上の結果から、スキルの修正による手順の固定化が再現性の高さに寄与していることが確認された。ただし、LLM の出力は本質的に確率的であり、サンプルサイズ（10 回）を超えて厳密に 100%の再現率が保証されるものではない点に留意する必要がある。

5.7.3　処理性能検証

本項では、LINKS BI システムの処理性能を実際の本番環境に対して計測した結果を記載する。データ容量・AI チャット連続処理・ダッシュボード搭載量・同時接続ユーザー数・プロンプトレイテンシの 5 項目を検証した。

検証の概要

本節では、LINKS BI システムの処理性能を実際の本番環境に対して計測した結果を記載する。検証はローカル Windows 11 Home Version（AMD Ryzen 9 6900HX / 32GB）から Playwright 経由で実施した。データ容量・AI チャット連続処理・ダッシュボード搭載量・同時接続ユーザー数・プロンプトレイテンシの 5 項目を、小規模（基本動作）→ 中規模（実運用想定）→ 大規模（限界値測定）の 3 段階で検証した。

テスト条件の「小規模・中規模・大規模」の 3 段階は以下の考え方で設定した。小規模は最小限の基本動作確認であり、日常的な利用シーンに相当する。中規模は実証事業で想定される実運用シナリオに相当する。大規模はシステムの限界値を測定するストレステストであり、将来的なデータ量増加や利用者拡大に備えた余裕度の確認を目的とする。

具体的な数値の根拠は以下のとおりである。B-1 のデータ容量 300MB は、実証事業で扱う最大規模のデータセット（ドローン飛行計画データ、約 100 万〜200 万レコード）を上回る規模として設定した。B-4 の同時接続 30 並列は、AI チャットの同時利用がシステム全体の同時接続ユーザーの一部と想定した場合の上限値である。

検証結果サマリ

各項目の合格基準は「エラーなく処理が完了すること」とした。具体的には、B-1 ではチャート生成が正常に完了すること、B-2 ではエラーなく連続応答が返ること、B-3 ではエクスポートしたウィジェットがリロード後に復元されること、B-4 では全リクエストに応答が返ること、B-5 ではグラフ描画が正常に完了することを確認した。応答時間の劣化は合否判定には含めず、傾向の記録として計測している。

表 5.9　処理性能検証結果サマリ

No.	評価指標	小規模	中規模	大規模
B-1	データ容量	1MB （展開 1.4MB）	10MB （展開 14MB）	300MB （展開 577MB）
B-2	AI チャット連続処理	3 メッセージ 平均 10.9s	8 メッセージ 平均 13.1s	15 メッセージ 平均 16.6s
B-3	ダッシュボード搭載量	3 件中 3 件復元	8 件中 7 件復元	15 件中 12 件復元
B-4	同時接続ユーザー数	3 並列　全件成功	10 並列 全件成功	30 並列 全件成功
B-5	プロンプトレイテンシ	簡単　平均 13.0s	中程度　平均 21.7s	複雑 平均 36.7s

B-1 データ容量

LINKS Tables から Parquet をインポートし、AI にチャート描画を依頼した際の成否と所要時間を計測した。

表 5.10　B-1　データ容量別の処理結果

規模	ファイルサイズ	展開後サイズ	行数	カラム数	インポート時間	チャート応答時間
小規模	1 MB	1.4 MB	30,000 行	10	4.0s	16.0s
中規模	10 MB	14 MB	200,000 行	16	4.1s	19.8s
大規模	300 MB	577 MB	8,000,000 行	35	28.7s	25.0s

800 万行・577MB の大規模データにおいても正常にチャート生成が完了した。Parquet の圧縮率は約 2 倍（300MB → 展開後 577MB）であり、DuckDB-Wasm によるブラウザ内分析処理

が大規模データに対しても有効に機能することが確認された。大規模データにおいては、インポートに 28.7 秒かかるが、これはサーバーから 300MB のデータをダウンロードする時間も含むことと、実用上 8,000,000 行のデータを扱うことは相当規模と考えられるため、実用上問題のない処理速度を達成していると考える。

B-2 AI チャット連続処理

サンプルデータ読み込み後、連続でグラフ生成を依頼した際のエラー発生有無と応答時間を計測した。これは、連続でチャットを継続した際に、途中でシステムがクラッシュするなどの問題が起きないか、また想定される連続のチャットに対して応答ができるかを検証するものである。

表 5.11　B-2 AI チャット連続処理の結果

規模	メッセージ数	成功数	エラー数	合計時間	平均応答時間
小規模	3	3	0	32.8s	10.9s
中規模	8	8	0	104.6s	13.1s
大規模	15	15	0	248.4s	16.6s

15 メッセージの連続処理においてもエラーは発生しなかった。メッセージ数増加に伴い平均応答時間が 10.9s→16.6s と微増する傾向が確認されたが、これはコンテキスト長の増加に起因する想定内の変動である。検証した 15 メッセージでは問題は見られなかったが、実際には長大なプロンプトをユーザーが入力することも想定されるため、チャット内でのトークン量が多くなった際の最適化処理の余地は残っている。

B-3 ダッシュボード搭載量

チャットでグラフ生成後、ダッシュボードにエクスポートし、リロード後のウィジェット復元を検証した。

表 5.12　B-3 ダッシュボード搭載量の検証結果

規模	ウィジェット数	チャット生成	エクスポート成功	リロード復元	リロード時間
小規模	3	3 件中 3 件成功	3 件中 3 件成功	3 件中 3 件復元	2.8s
中規模	8	8 件中 8 件成功	8 件中 7 件成功	7 件中 7 件復元	3.2s
大規模	15	15 件中 15 件成功	15 件中 12 件成功	12 件中 12 件復元	3.5s

大規模（15 ウィジェット）ではエクスポート成功率が 80%（12/15）となり、一部エクスポート失敗が発生した。復元したウィジェットはすべて正常に表示されており、リロード時間は 3.5 秒程度で安定していた。

B-4 同時接続ユーザー数

単一マシンから複数ブラウザコンテキストで同時に AI チャットリクエストを送信し、応答時間の劣化を計測した。

表 5.13　B-4 同時接続ユーザー数の検証結果

並列数	成功数	合計時間	平均応答時間	最小	最大
3	3 件中 3 件成功	18.0s	17.6s	17.0s	18.0s
10	10 件中 10 件成功	29.1s	23.4s	18.1s	29.1s
30	30 件中 30 件成功	102.5s	88.7s	56.4s	102.5s

30 並列まで全件成功した。ただし並列数に応じて応答時間が劣化する傾向が確認された。3 並列では単体と同等（17〜18s）だが、10 並列では平均 23.4s、30 並列では平均 88.7s に増加し、ばらつきも大きくなった（56〜103s）。これは LLM API のレートリミット制約が主因と考えられる。現状、同時接続数が増えてもチャットが止まるという事象は観測されていないが、同時に処理できるトークン上限は AWS Bedorock のレートリミットに依存するため、レートリミットが緩和するまで一時的な待ちなどが発生する可能性がある。

B-5 プロンプトレイテンシ

プロンプトの複雑さ別に応答時間を計測した（各 3 回実行）。

表 59　B-5 プロンプトレイテンシの計測結果

プロンプト種別	内容	平均	最小	最大	P50	P95
簡単	サンプルデータで棒グラフを表示	13.0s	13.0s	13.1s	13.0s	13.1s
中程度	カテゴリ別集計＋ 上位 10 件グラフ表示	21.7s	21.0s	22.1s	22.0s	22.1s
複雑	ヒストグラム＋ カテゴリ別棒グラフ表示	36.7s	36.1s	37.1s	37.1s	37.1s

※P50：中央値。P95：95 パーセンタイル（全計測値の 95%がこの値以下に収まる）。

簡単〜複雑なプロンプトで約 24 秒の差があり、プロンプトの複雑さに比例して応答時間が増加することが確認された。いずれのプロンプトでもグラフ描画の検証は全て合格（エラーなし）しており、実用上の問題はないと判断した。

考察

本検証を通じて、データ活用モジュール（LINKS BI）は実運用を想定した処理負荷に対して十分な性能を有することが確認された。特に大規模データ（300MB・800 万行）に対するチャー

ト生成、15 メッセージの連続処理、30 ユーザーの同時接続がいずれも成功しており、実証フェーズでの運用基盤として機能することが示された。

一方で、各テスト項目にはそれぞれ固有のボトルネックが存在する。以下にテスト項目ごとのボトルネックと対処の方向性を整理する。

- B-1（データ容量）：ボトルネックは DuckDB-Wasm のブラウザ内メモリである。WebAssembly のメモリ空間は仕様上 4GB が上限であり、展開後のデータサイズがこの制約に近づくと処理が不安定になる可能性がある。今回の検証では 577MB の展開データに対して正常動作を確認しており、現時点では十分な余裕がある。将来的により大規模なデータを扱う場合は、Parquet の列・行グループ単位での部分取得（5.1 節参照）や事前集約による行数削減で対処できる。
- B-2（連続処理）：ボトルネックはコンテキストウィンドウの蓄積による LLM の応答時間増加である。メッセージ数が増えるほど会話履歴がトークンを消費し、AI の思考時間が伸びる。5.9.2 項で述べるコンテキストコンパクション（過去の会話履歴の要約・圧縮）により改善が期待される。
- B-3（ダッシュボード搭載量）：15 ウィジェット中 3 件のエクスポート失敗が確認された。復元したウィジェットはすべて正常に表示されているため、エクスポート処理自体の安定化が今後の課題である。
- B-4（同時接続）：現在は AWS Bedrock のクロスリージョン推論（日本）を利用し、東京・大阪の 2 リージョンに自動分散されているため、現時点では Bedrock のクォータ（1 分あたりのリクエスト数・トークン数の上限）に余裕があるものの、将来的により利用者数が拡大すると、この上限が制約となる可能性がある。クォータの引き上げ申請やプロビジョンドスループット（Provisioned Throughput）の活用が今後の対処策となる。
- B-5（プロンプトレイテンシ）：プロンプトの複雑さに比例して応答時間が増加する傾向が確認されたが、いずれのケースでもエラーなく処理が完了しており、現時点で対処は不要と判断した。

5.8　得られた知見

AI チャット機能の開発・チューニングを通じて、LLM の動作が不安定になる典型的なパターンとその対処法を体系化した。以下の 4 つのパターンを特定している。

5.8.1　パターン 1：数値指標が実行ごとにブレる

同一のプロンプトに対して AI の出力が実行ごとに異なる問題である。たとえば集計結果のカラム名が実行ごとに変わる、ソート順序が一定しない、集計方法（SUM vs COUNT）の選択が揺れるといった事象が発生する。

対処法として、スキルファイルに MUST/NEVER ルールで手順を固定する方式が有効であった。「集計結果のカラム名は元のカラム名をそのまま使用すること（MUST）」「独自のエイリアスを付与しないこと（NEVER）」のように、具体的な制約を明示することでブレが解消された。特に、正しいパターンと誤ったパターンの両方を例示する方式が効果的であった。

5.8.2　パターン 2：AI がスキルを取得しない

ユーザーの指示に関連するスキルが存在するにもかかわらず、AI が 「get_skill」 を呼び出さずに処理を進めてしまう問題である。この結果、スキルに記載された手順やパターンが適用されず、品質の低い出力が生成される。

対処法として、2 つのアプローチを組み合わせている。第 1 に、スキルファイルの 「tasks」 キーワードが実際のユーザー指示で使われる用語と一致しているかを確認する。日本語と英語の両方のキーワードを網羅的に登録することが重要である。第 2 に、「deps」 フィールドによる依存スキルの連鎖取得を活用する。基盤となるスキルを 「deps」 に指定しておくと、関連スキルが自動的に co-fetch されるため、AI が個別にスキルを判断する負荷が軽減される。

5.8.3　パターン 3：ガイドライン・プロンプトの矛盾と曖昧さ

システムプロンプト内の異なるセクション間、またはシステムプロンプトとスキルファイル間で指示が矛盾している場合に、AI の動作が不安定になる問題である。たとえば「常にテーブル構造を確認してからクエリを実行せよ」という指示と「テーブル探索ステップを省略して直接クエリを実行せよ」という指示が共存すると、AI の判断が実行ごとに揺れる。

対処法として、ルールの集約と CRITICAL/IMPORTANT 階層の厳格な運用が有効であった。同一の話題に関するルールは 1 箇所に集約し、プロンプト内の散在を排除する。また、5.2.4 項で述べた強調キーワードの階層構造を厳格に運用し、最上位の RULE ZERO のみに CRITICAL を使用することで、AI が優先順位を正しく判断できるようにする。

5.8.4　パターン 4：AI が途中で停止する

複数ステップの処理を実行中に AI がテキスト応答を返して処理を中断する問題である。特にコンテキストウィンドウが肥大化した状態で発生しやすい。

対処法として、スキルファイルの 200 行上限を厳守し、超過するスキルは分割する。また、テキスト検索のレスポンスサイズを制限してコンテキストの肥大化を防ぐ。5.5 節で述べた国会答弁案生成の最適化では、テキスト検索レスポンスを 86,000 文字から 11,000 文字に制限したことで、コンテキスト肥大化に起因する処理停止が解消された。

5.9　課題と今後の方針

本章では、DuckDB-Wasm によるブラウザ内分析基盤、エージェンティック AI アーキテクチャとスキルシステム、国会答弁案生成の性能最適化など、LINKS BI の技術的工夫を解説した。精度検証では 20 シナリオ全件で 100%の再現率を達成し、処理性能検証では 300MB・800 万行の大規模データや 30 並列の同時接続に対しても安定動作することを確認した。これらの結果から、データ活用モジュール（LINKS BI）は実証フェーズのシステムとして十分な品質と性能を有すると判断できる。

一方で、実用化に向けてはいくつかの課題が残されている。以下にそれぞれの課題と今後の対応方針を整理する。

5.9.1　サブエージェントの導入

現在の LINKS BI の AI チャットは単一の AI エージェントがすべての処理を実行する設計である。国会答弁案生成のような複雑なタスクでは、検索・構成・執筆・検証といった異なる性質の処理を 1 つのエージェントが順次実行するため、コンテキストウィンドウの圧迫と処理時間の増大が課題となっている。今後は、タスクの性質に応じて専門化したサブエージェント（検索エージェント・執筆エージェント・検証エージェント等）に処理を分担させるアーキテクチャの導入を検討する。

5.9.2　コンテキストのコンパクション

長い会話セッションではコンテキストウィンドウが蓄積された会話履歴で圧迫され、AI の応答品質が低下する。Sonnet 4.6 等の新しいモデルではモデルレベルのコンパクション（過去の会話履歴を要約して圧縮する機能）が期待されており、これを活用したコンテキスト管理の最適化を計画している。

5.9.3　スキル検索の高度化

現在のスキル選択は AI がスキルカタログ（各スキルの名前と説明の一覧）を参照して自律的に判断する方式である。このためスキル数が増加するとカタログ自体がコンテキストウィンドウを圧迫し、AI のスキル選択精度が低下する懸念がある。この問題に対しては、スキルカタログの階層化（カテゴリ→サブカテゴリ→個別スキルの段階的絞り込み）、説明文の意味的類似度に基づくセマンティック検索の導入、スキル選択を専門に行うサブエージェントの活用などを検討している。

5.9.4　逆質問（プランニングモード）機能

現在の AI チャットはユーザーの指示を受け取ると即座に処理を開始するため、曖昧な指示に対して AI が独自に解釈し結果がブレる問題がある（6.3.1 節参照）。この課題に対して、AI が処理実行前にユーザーへ逆質問して解釈の前提（集計単位、NULL の扱い等）を確認するプランニングモードの導入を検討している。

5.9.5　ツール説明文の改善

Function calling におけるツールの説明文は、AI がどのツールを選択するかの判断に直接影響する。5.5 節で述べたように、説明文の長さや記述内容によって AI のツール選択に偏りが生じる問題が確認されている。ツールの粒度（1 つのツールが担う機能の範囲）の見直しを含め、AI が正確にツールを選択できるよう改善を検討する。

第 6 章　実証結果・有用性検証

本章では、 LINKS Veda の実証実験を通じて得られた有用性の評価を記録する。技術的な詳細は第 4 章（データ構造化）・第 5 章（LINKS BI）に委ね、本章はユーザー視点の体験評価に焦点を当てる。

6.1　評価の観点と方法

実証にはアンケートも実施したが、設問数や回答者数の制約から集計結果のみで判断することは難しい。そのため本章の評価は、実証現場に立ち会った開発・運営チームが直接観察した利用状況と、ユーザーとのやり取りから得た印象を総括的に判断したものを中心に記述する。
「評価が高かった機能」と「実用上厳しいと感じた機能」について、なぜそうなったかの原因分析を含めて整理することを重視している。なお、利用ログの自動取得によるユーザー利用実態の把握（どの画面で躓いているなどの分析）は今後の課題であり、現時点では実証の中で得られた観察ベースの評価を主軸としている。

6.1.1　評価の枠組み

本章の評価は以下の 2 軸で整理する。

- 有用と評価されたもの：実証の中でユーザーが詰まらずに操作でき、「使える」と判断した機能・体験。実証立会い者が観察した反応（手が止まらない、説明なしに進める、自然に使い続ける等）を根拠とする。
- 明らかになった課題：実証立会い者が「これは業務では使いにくい」「ここが障壁になっている」と判断した機能・体験。ユーザーの発言・操作の詰まり方・途中離脱の発生などから判断したものを含む。各課題については、なぜその問題が起きているかの原因分析もあわせて記述する。

評価はデータ構築モジュール（6.2 節）とデータ活用モジュール（LINKS BI）（6.3 節）の 2 モジュールに分けて整理する。

6.2　データ構築モジュールの有用性評価

6.2.1　構造化フローの体験評価

本項では、PDF のアップロードからスキーマサジェッション・構造化処理・結果確認・BI への連携までの一連のフローについて、ユーザーが一気通貫で操作を完了できるかを評価する。

有用と評価されたもの

（1）初めてのユーザーでも構造化フローを完了できた
2024 年度は「ワークフロー」という抽象的な概念をユーザーが理解し、処理ステップを自ら構成する必要があり、途中で操作を諦めるユーザーが多かった（第 3 章 3.3.1 節参照）。2025 年度は、このワークフロー実行までの流れを、一連のフォーム形式のステップに再設計し、スキ

ーマサジェッション→構造化処理→結果確認をボタンを順に押すだけで完了できる設計とした。実証では初めてシステムを触ったユーザーでも途中で詰まらずフロー全体を完了でき、操作途中での離脱が大幅に減少した。

利用者の声：

- 「データをドラッグするだけで構造化でき、様々な事業に活用できる」（国交省部門職員）
- 「データを読み込ませると、システムからどういう構造にできそうかを提案してもらえるフローがあると実務に非常に活用しやすい」（国交省部門職員）

スキーマサジェッション機能の価値が複数の業務を横断して確認された。

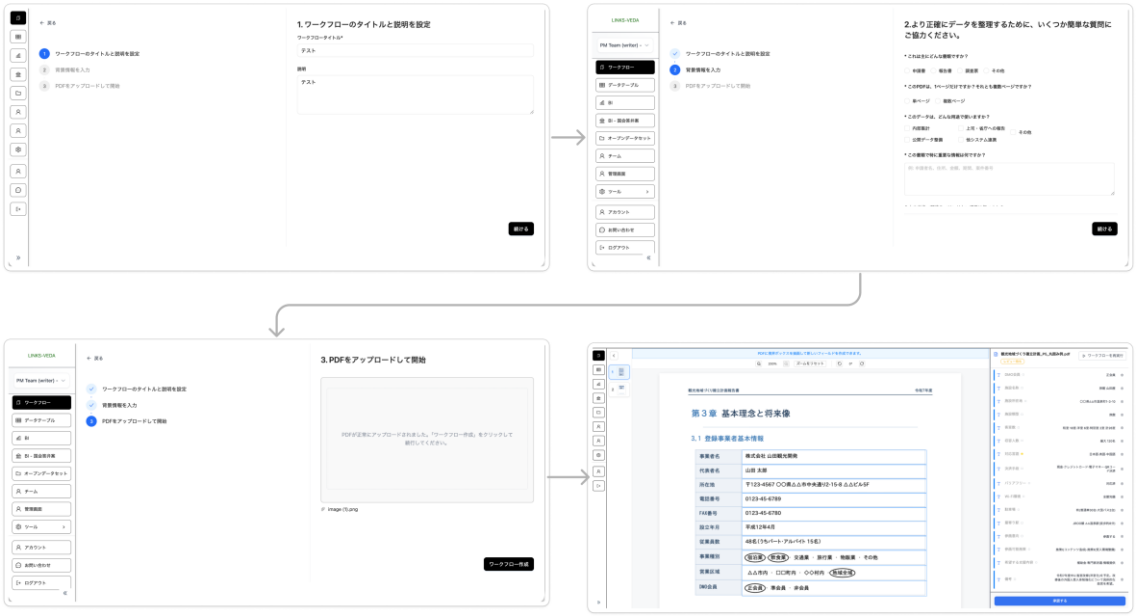


図 45 構造化までの一連の流れ

（2）データ構築モジュールから BI へのシームレスな連携体験

構造化したデータを BI に取り込み、チャートや地図として可視化するまでのフローを、ユーザーが技術的な操作をほとんど意識することなく体験できた。非構造データから可視化までが一貫したシステムで完結する点は、今年度のシステム設計の中核的な提供価値であり、実証を通じてその有用性が確認された。

利用者の声：

- 「PDF をシステムにアップロードするだけで最終的に地図や表で見られるのは魔法のようだ」（国交省部門職員）

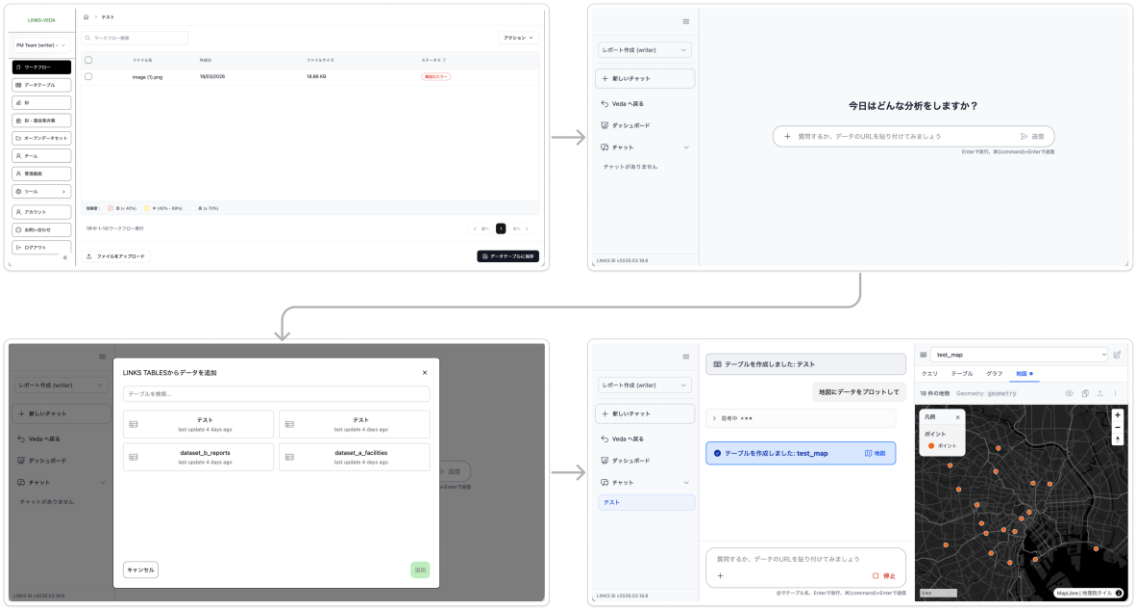


図 46 Veda から BI への連携体験

（3）紙ベース業務における活用への期待

既存の紙ベース業務を抱える分野では、構造化フロー自体に対する高い期待が示された。

利用者の声：

- 「現状は、紙の届け出を担当者が手入力し、エクセルに転記しており、こういった業務にこそ活用できれば、データを打ち込む手間がなくなり現場にとっても非常に嬉しい」（国交省部門職員）

構造化フローが紙帳票処理の自動化手段として直接的に評価された形である。

明らかになった課題

（1）様式ブレへの対応限界と部分的再構造化の不足

行政データには様式が完全に統一されていない PDF が多く存在する。フォントサイズの違い・表のレイアウトのズレ・手書き記入の混在等により、同じスキーマ定義でも抽出精度にバラツキが生じた。省内研修会アンケート（n=17）では「設定条件のわかりやすさ」について 65%が肯定的である一方、否定的評価も 2 件あった。

加えて、抽出精度が低かった箇所に対して「特定のフィールドのみを対象に再構造化する」「PDF の特定領域を指定して再試行する」といった部分的・柔軟な再処理ができない点が課題として確認された。

利用者の声：

- 「赤や黄による信頼度表示は有効だが、色が付いていない箇所にも誤りがないとは限らないため、利用時には注意しなければならないと思う」（国交省部門職員）
- 「ミスが許されにくい領域では、結果をそのまま使用するのではなく、補助的に活用する姿勢が現実的」（国交省部門職員）
- 「PDF 以外からもデータ読み取りができることを期待したい」（国交省部門職員）

今後は抽出精度が低い箇所や、抽出自体に失敗した箇所にピンポイントで部分的な構造化の再実行の仕組みを検討する。

（2）大量ファイル処理時の処理時間

省内研修会アンケート（n=17）では「レスポンス速度」の評価が最も低く、否定的評価が 59%（10 件）に達した。研修会という同時多数アクセス環境での負荷集中が主因と推測されるが、個別検証会でも処理時間への指摘が確認されている。

利用者の声：

- 「動作は遅いと感じた」「システムには、より安定した操作体験や挙動を提供してほしい」（国交省部門職員）
- 「実行時間が長い部分があり、データ作成および分析に時間を要する」（国交省部門職員）
- 「業務終了時に条件設定を行っておけば翌朝には分析結果が出力されているような運用が望ましい」（国交省部門職員）

処理時間の改善に加え、長時間を要する処理のスケジュール実行機能の導入も検討課題として挙がった。

6.2.2　構造化結果の品質と確認体験

本項では、AI 処理の不確実性（ハルシネーション・誤抽出）に対してシステムがユーザーを適切に支援できるかを評価する。

有用と評価されたもの

（1）信頼度スコアとバウンディングボックスによる確認支援

構造化結果の品質確認において、2 つの機能が有用と評価された。第 1 に、LLM による構造化処理の結果に対して信頼度スコアを付与し、信頼度の低いフィールドを UI 上でハイライト表示する機能である。ユーザーはすべてのフィールドを目視確認するのではなく、ハイライトされた箇所を重点的に確認・修正できる。第 2 に、各フィールドの抽出値が PDF 上のどの箇所から取得されたかをバウンディングボックス（矩形ハイライト）で視覚的に示す機能である。元の PDF の「どの領域」から「どの値」が取り出されたかが一目で確認でき、2024 年度に必要だった PDF を別途開いての目視照合が不要になった。

利用者の声：

- 「AI の出力結果をそのまま信頼するのではなく、どこを確認すべきかが分かる点が重要」（国交省部門職員）
- 「信頼性が低いような数値を教えてもらえるところが非常に助かる」（国交省部門職員）

情報の正確性を重視する国交省職員にとって、AI の出力を検証するための支援機能として高く評価された。

明らかになった課題

（1）構造化精度の限界と現実的な活用方針

現時点での構造化精度でも利用価値があると考える利用者がいる一方で、様式のバリエーションごとの個別チューニングが必要なケースがあり、精度に対する要求が高い利用者の要望に全ては応えられなかった。

利用者の声：

- 「全件を人の手で検証することは現実的ではなく、概況把握や一次分析用途としては十分実用的」（国交省部門職員）
- 「実績報告データには事業者による重複提出等が見られ、捕捉率が実態を正確に反映していない可能性がある」（国交省部門職員）

後者の指摘は、構造化精度以前に元データの品質がボトルネックとなるケースもあることを示している。精度の全件検証は非現実的であるため、概況把握・一次分析用途としての実用性を前提に、信頼度スコアによる要確認箇所の重点検証と、既存業務の上に補助的活用として組み合わせる運用が現実的なアプローチとして整理された。

6.2.3　データ加工機能の評価

本項では、構造化されたデータを業務で活用可能な状態に加工・整備するための機能が十分に機能したかを評価する。

有用と評価されたもの

（1）標準的なテーブルアクションとデータクレンジング

型変換・文字列正規化・全角半角統一・住所正規化・ジオコーディングといった行政データ整備に頻出する処理を、コードを書かずに UI から実行できる点は実証で有用性が確認された。特にジオコーディング（住所→緯度経度変換）は BI での地図可視化に直結するため、構造化→ジオコーディング→地図表示の一連のフローとして体験できた点の評価が高かった。

実証会では、様々なデータをこれらのテーブルアクションを利用してノーコードで処理したい、との声があがり、実業務で利用を検討する職員も見られた。

利用者の声：

- 「データの結合率や信頼度など最終確認を経ることで、各課でも十分に活用できる内容」（国交省部門職員）
- 「複数のデータを統合したときに異常値が分かるような提案やその抽出ができれば、よりよいツールになる」（国交省部門職員）

明らかになった課題

（1）テーブルアクションの再利用性と Excel との機能差

構造化結果データを BI での分析ができる状態にするには、複数のデータ加工処理を組み合わせる必要があるが、現状はそれらを再利用可能なパイプラインとして保存・管理する機能が不足している。ある部門では、データを集計して資料化する業務を毎月行うなど定型業務があるため、これらを少しでも効率化したいとの声があった。

加えて、ユーザーの比較対象は Excel であり、カラムのユニーク値確認・値の分布把握・加工後のデータ状態の確認・複数アクション適用後の中間状態の追跡といった操作が不足している点が課題として挙がった。

今後は、複数のテーブルアクションをパイプラインとして保存・再利用できる機能の実装と、データ探索機能（ユニーク値一覧・分布ヒストグラム・中間状態プレビュー等）の強化を検討する。

6.3　データ活用モジュール（LINKS BI）の有用性評価

6.3.1　AI チャットによる可視化・データ分析

本観点では、非技術者ユーザーが自然言語でデータに対して質問・可視化指示を行い、期待する結果を得られるかを評価する。

有用と評価されたもの

（1）チャットベースの直感的な可視化操作

自然言語の指示だけで AI が適切な SQL クエリを生成・実行し、グラフや地図を描画するフローが実証で機能した。2024 年度にユースケースごとの個別アプリケーションに依存していた体験を、統一的なチャットインターフェースに集約した改善方針の正しさが確認された。棒グラフ・折れ線グラフ・散布図・ヒートマップ・地図（点・面・メッシュ）・テーブル等、多様な可視化形式を AI が状況に応じて選択・生成する能力も実証された。

利用者の声：

- 「自分では作成が難しい分析表が簡単に作成できる」（国交省部門職員）
- 「半角・全角が混ざっていても拾ってくれる」「口語体でも対応してくれるのは使いやすい」（国交省部門職員）

カラム名や集計方法の知識を持たない国交省職員でも、自然な言葉で操作できる設計が実際の利便性につながっていることが確認された。

（2）業務効率化・資料作成補助としての実用性

AI チャットによる即時の集計・可視化が、日常的な資料作成業務の効率化手段として機能することへの期待が複数の業務にわたって確認された。

利用者の声：

- 「短時間で集計やグラフ化ができる」「内部資料や統計資料の作成において作業負担の軽減につながる」（国交省部門職員）
- 「これまで Excel でアナログ的に行っていた集計作業が抽出できる点は大変便利な機能」（国交省部門職員）
- 「宝の持ち腐れだったデータが、今後の政策立案に大いに活かせると感じた」（国交省部門職員）

特に、これまで紙ベースで保管され提出件数の把握程度にしか使われていなかった事業報告書・実績報告書が容易に分析できるようになった点が評価された。一方で、BI の出力をそのまま成果物とするのではなく、Excel で作成した集計結果との突合・妥当性確認に活用するという既存業務フローとの併用が、現実的なアプローチとして整理された。

明らかになった課題

（1）AI 出力の再現性・安定性の問題

同一のプロンプトと同一のデータに対しても、試行ごとに異なる集計結果や可視化が生成されるケースがあった。原因は、NULL の扱い・集計単位・配列データの展開深度等、明示されな

い前提条件に対する AI の解釈が確率的にブレることにある。AI がユーザーに追加質問を行い、あいまいな前提を事前に解消するプランニングモードの導入が検討される。

（2）AI 処理の透明性・追跡可能性の不足

AI が裏側で生成・実行する SQL 処理がユーザーに見えないため、「表示された数字が正しいかどうかを確認したい」というニーズに応えられていない。同様に、ダッシュボードに保存されたウィジェットについて「このグラフはどのデータを使って何を集計したものか」を事後的に確認する手段も不十分である。

利用者の声：

- 「ダッシュボードがどのように作られたか逆引きしたい」（国交省部門職員）

改善策として、実行された SQL を平易な文章に変換して提示する機能、集計処理のステップを追跡できるモード、ウィジェットへのメタ情報付与（使用テーブル・実行クエリ・生成日時・使用プロンプト等）の導入が検討される。

（3）データ理解の支援不足

テーブル構造の理解が不十分なまま操作した場合に誤ったグラフが生成されるリスクが指摘された。現状の UI は「ユーザーがデータ構造を把握していること」を前提としており、データ理解を能動的に支援する機能が不足している。

利用者の声：

- 「BI を使用する人が元のテーブルのカラム情報をよく理解していないといけない」（国交省部門職員）
- 「テーブルを指定せずに、BI との会話で作成したいグラフの指示から BI がテーブルを提示してくれる機能が望ましい」（国交省部門職員）
- 「チャット側から、このような分析ができるなどの提案があるといったものを検討できないか」（国交省部門職員）

改善策として、チャット内でのテーブル構造・カラム定義の自動提示、データロード時の自動プロファイリング、分析メニューの提示機能の導入が検討される。

（4）操作習熟のための学習コスト

自然言語での操作は直感的ではあるが、効果的な指示の出し方には一定の学習が伴うことが確認された。

利用者の声：

- 「チャット指示の出し方に慣れが必要」「コツをつかめば出したい結果を出せる」（国交省部門職員）

データロード時に分析例をサジェストする機能や、「このデータでできる分析」をパネル表示するガイド機能の導入により、習熟コストの低減を検討する。

6.3.2　特化型 AI 機能（国会答弁案生成等）

本観点では、汎用データ分析を超えた行政業務特有のユースケースに特化した AI 機能の有用性を評価する。

有用と評価されたもの

（1）国会答弁案生成機能の有用性

過去の国会答弁データを DuckDB-Wasm で検索・参照しながら、新たな質問に対する答弁案を自動生成する機能は、実証で最も高い評価を得た機能の一つである。単なる LLM による文章生成と比較して評価された点は 2 点ある。第 1 は、生成された答弁案の各段落が「どの過去答弁を参照して記述されたか」の引用根拠が明示される点である。第 2 は、答弁案の文字数・段落数等の様式要件を自動バリデーションし、基準を満たすまで再生成するループが自動で動作する点である。

利用者の声：

- 「かなりクオリティの高い答弁が生成されており驚いた」「段落ごとにそれらしい内容がきちんと表示されており完成度が高い」（国交省部門職員）
- 「国会答弁は時間勝負の中で、こういったものが自動で生成できると非常に有意義ではないか」（国交省部門職員）
- 「担当局が内容の正誤を最終確認する必要はあるものの、使い方を間違わなければ各局でも十分に活用してもらえる内容」（国交省部門職員）

根拠の追跡可能性と時間的制約の厳しい答弁業務での効率化効果が評価され、最終確認を前提とした補助ツールとしての省内展開可能性が示唆された。

明らかになった課題

（1）参照データの品質・範囲への依存

AI が参照できる情報はシステムに取り込まれたデータに限られるため、取り込まれていない最新情報や関連省庁の情報等には対応できない。生成結果の品質は入力データの品質と範囲に直接依存しており、データ整備の工数が成果物の品質に大きく影響する。 例として、「次期アメリカ大統領は誰だと思うか」のような国会答弁と無関係の問いを入力した場合、システムは答弁案の生成を行わず、「お尋ねの内容は、国土交通行政に関する質問ではなく、また『国会答弁書データ』に該当する素材が存在しないため、答弁案を作成することができません」と応答する。あわせて、国会答弁案作成モードが対応する条件として、国土交通行政に関連する質問であること、過去の国会答弁データベースに類似の答弁事例が存在すること、省庁としての公式見解を示すべき政策課題であることの 3 点が提示される。このように、適用範囲外の問いに対してはシステム自身が理由を説明した上で生成を拒否する動作となっている。

6.3.3　システムの安定性・スケーラビリティ

本観点では、複数ユーザーが同時に利用する実運用環境におけるシステムの安定性・応答性を評価する。

有用と評価されたもの

（1）ブラウザ内処理によるスケーラビリティの実現

DuckDB-Wasm によるブラウザ内処理の採用（5.1 節参照）により、2024 年度のサーバー経由処理で発生していた応答遅延が解消され、100 万～200 万行規模の大規模データに対する集計・可視化の応答速度が実用レベルに改善した。ユーザー数が増加してもサーバー側の負荷が増大しないスケーラブルな構成となっている。

明らかになった課題

（1）長時間セッションにおける AI 応答品質の低下

複雑な分析を繰り返す長時間のセッションでは、後半になるほど AI の応答が遅くなり、応答内容の精度も前半と比べて不安定になる場合があることが確認された。これは会話履歴の蓄積によるコンテキストウィンドウの圧迫に起因するものであり（5.7.3 項の処理性能検証 B-2 参照）、コンテキストコンパクションの導入により改善を図る予定である（5.9.2 項参照）。

（2）ブラウザ内処理におけるメモリ上限の存在

DuckDB-Wasm のブラウザ内処理には WebAssembly のメモリ上限（4GB）という構造的な制約がある。現時点では十分な余裕があるが、将来的なデータ規模の拡大に備えた対処策の検討が必要である（詳細は 5.7.3 項の処理性能検証 B-1 参照）。

6.4　本章のまとめ

本章では、Veda の実証を通じて得られたユーザー視点の評価を整理した。両モジュールとも、2025 年度の再設計により UI/UX と基本機能の有用性が確認された一方、テーブルアクションの再利用性、AI 出力の再現性、処理速度等の課題も明らかになった。各課題に対する対応方針は本章の各節に記載しており、これらを踏まえた成果・課題の総括は第 8 章で行う。

第 7 章　ランニングコスト

本章では、 Veda のインフラストラクチャ運用に要した費用の実績を記録する。クラウドプロバイダー別（Google Cloud・AWS・Azure）にサービス別コスト内訳を整理し、社会実装・継続運用における費用感を示す。対象期間は 2025 年 4 月～2026 年 2 月である。なお、本章の金額は本番環境と開発環境の合算値として記載する。

7.1　Google Cloud のコスト

本節では Google Cloud のコスト実績を整理する。対象期間は 2025 年 4 月～2026 年 3 月初旬である。金額は JPY（日本円）で計上されており通貨換算は発生していない。
累計コストは¥3,366,447 である。本番環境と開発環境を合算したサービス別の内訳を以下に示す。

表 7.1　Google Cloud サービス別コスト内訳

サービス	製品・概要	金額（税込）
Cloud SQL	PostgreSQL データベース（マネージドサービス）	¥1,538,495
Compute Engine	MongoDB セルフホスティング等	¥951,149
Cloud Run	API サーバー・バックエンドサービス・ 非同期処理実行基盤	¥543,447
Networking	Cloud Load Balancer・ネットワーク転送料	¥148,342
Cloud Monitoring	メトリクス収集・アラート設定	¥113,718
Vertex AI	LLM API（開発環境での検証利用）	¥24,047
Artifact Registry	コンテナイメージ管理	¥18,253
Cloud Logging	ログ収集・保管	¥8,650
その他	Secret Manager 等	¥20,346
合計		¥3,366,447

7.2　AWS のコスト（Bedrock）

本節では AWS Bedrock のコスト実績を整理する。対象期間は 2025 年 4 月～2026 年 3 月初旬である。データ構築モジュールのデータ構造化処理および LINKS BI の AI チャット機能は AWS Bedrock 上の Claude モデルを利用しており、Bedrock の課金は Google Cloud とは独立した AWS アカウントで管理されている。

累計コストは¥22,783,655 である。用途別の内訳を以下に示す。

表 7.2　AWS Bedrock 用途別コスト内訳

用途	製品・概要	金額（税込）
BI における AI チャット		¥4,971,156
Veda におけるデータ構造化	データ構築モジュールのデータ構造化処理（Claude Sonnet 4.5 / Haiku 4.5 等）	¥17,812,499
合計		¥22,783,655

7.3　Azure のコスト

本節では Microsoft Azure のコスト実績を整理する。データ構築モジュールのデータ構造化処理では、OCR（光学文字認識）の前段工程として Azure Document Intelligence を利用している。対象期間は、2025 年 4 月～2026 年 3 月初旬である。

表 7.3　Azure サービス別コスト内訳

サービス	用途	金額（税込）
Azure Document Intelligence	OCR・テキスト・レイアウト情報抽出	¥2,190,554

7.4　コスト総計

クラウドプロバイダー別のコスト総計を以下に示す。

表 7.4　コスト総計

クラウドプロバイダー	累計金額（税込）
Google Cloud	¥3,366,447
AWS（Bedrock）	¥22,783,655
Azure	¥2,190,554
総計	**¥28,340,656**

7.5　コスト構造の考察

前節までのコスト実績データに基づき、支出構造の特徴と最適化の方向性を考察する。

7.5.1　コスト構造の全体像

コスト総計¥28,340,656 のうち、AWS Bedrock（LLM API 利用料）が¥22,783,655（約 80%）を占めており、LLM コストが全体を支配する偏重構造となっている。Bedrock コストの内訳では、データ構造化処理が¥17,812,499（Bedrock 全体の約 78%）、BI の AI チャット機能が

¥4,971,156（約 22%）である。データ構造化処理は対象ファイル数に比例してコストが増大し、AI チャット機能はチャット 1 回あたりに複数の Function calling が発生するため累積コストが大きくなりやすい。現在利用しているモデルは Claude Sonnet 4.5 であり、今後データ処理量が増加するにつれコストはさらに拡大することが見込まれる。
Google Cloud（¥3,366,447、約 12%）では Cloud SQL（PostgreSQL）が最大のコスト要因であり、Google Cloud コストの約 46%を占める。Cloud SQL はマネージドサービスとして高可用性（HA）構成を採用しており、スタンバイインスタンスが常時起動していることが費用増加の主因である。次いで Compute Engine（約 28%）が大きいが、これは MongoDB のセルフホスティングに利用していたレガシーインスタンスであり、現在はデータを PostgreSQL に移管済みで使用していないため、停止により¥951,149 のコスト削減が見込める。
Azure（¥2,190,554、約 8%）は Document Intelligence（OCR 処理）の従量課金であり、日次の利用量により数百円〜十数万円規模で変動する。行政文書の大量処理タイミングと複数ユースケースの同日並列実行が重なった日には、1 日あたり約 18〜20 万円の課金が発生した事例も確認された。

7.5.2　モニタリングコストの最適化実績

前年度（2024 年度）では Cloud Monitoring のコストが Google Cloud 全体の大部分を占めていたが、メトリクス収集頻度の見直しにより改善された。本年度の Cloud Monitoring は累計¥113,718（Google Cloud コストの約 3%）に留まっており、最適化の効果が確認できる。

7.5.3　次年度の課題と打ち手

コスト最適化に向けて、以下の複数アプローチを次年度の課題として位置づける。

- モデルの使い分け：すべての処理に Sonnet を使用するのではなく、単純な分類・抽出タスクには Claude Haiku などの LLM を使うことで、精度を維持しつつ処理単価を抑えることができる可能性がある。
- プロンプトキャッシングの活用：Anthropic API はプロンプトキャッシング機能を提供しており、同一のシステムプロンプトやスキーマ定義が繰り返し送信される処理に対して入力トークンコストを大幅に削減できる。現在もプロンプトキャッシングは有効化しているが、最適化の余地がある。
- プロンプト・コンテキストの精査：スキーマ定義・スキルファイル・会話履歴等が必要以上のトークン量で LLM に渡されていないかを定期的にレビューし、不要なコンテキストを除去する。
- OSS LLM の自前ホスティング：機密性の高いデータや大量の定型処理に対しては、OSS の LLM を自前ホスティングし従量課金 API への依存を減らす選択肢もある。インスタンスのコストや運用に伴う人的コストと比較して検討する必要がある。
- Google Cloud のインスタンス最適化：使用していない Compute Engine インスタンスの停止、および Cloud SQL のインスタンスサイズの精査により、Google Cloud コストの削減余地がある。

7.6　人的コスト

本節では、本実証に要した人的コストを「インフラストラクチャ管理」と「ユースケース別データ構造化作業」の 2 つに分けて記載する。

7.6.1　インフラストラクチャ管理

本システムの運用基盤の構築・維持に要した工数は年間で約 10 人月である。主な作業内容は以下のとおりである。

- Terraform によるインフラストラクチャのコード化（IaC）の構築・管理：Google Cloud・AWS・Azure の各リソースをコードで定義し、環境の再現性と変更管理を確保する。
- モニタリング・アラート設定：Cloud Monitoring・Cloud Logging の設定、異常検知アラートの構築と運用。
- Web アプリケーションファイアウォール（WAF）の設定：Cloud Armor 等を用いたセキュリティポリシーの設定・チューニング。
- CI/CD パイプラインの整備：Artifact Registry を用いたコンテナイメージ管理、デプロイ自動化の構築。
- ネットワーク・認証基盤の管理：Cloud Load Balancer、Cloud Identity Platform、SSL 証明書等の設定・維持。

7.6.2　データ構造化作業

有用性検証として実施したユースケースのうち作業種類別に必要となった平均人的コストを以下に記載する。ユースケースごとの構造化処理にかかる作業種別は、以下の表 7.5 で示すとおりである。紙スキャンとは、紙で保管されている帳票をスキャナーで PDF 変換することを指す。構造化処理は、そうした PDF データや、Excel などの電子データを入力として、データ構造の設定からデータ構造化処理完了までにかかる作業を指す。クレンジング処理はデータ構造化処理によって出力されたデータを用いた、正規化処理や結合処理その他テーブルアクションを実施するまでの作業を指す。分析に適したデータ構造の検討や Veda への入力データを分類する事前作業などは含まない。表 7.6 はそれぞれの作業種別において、平均的に各工程にどの程度の人月工数がかかるかを記載したものである。本年度は、地域を限定して実証したケースも多かったため、概ね記載のとおりである。紙データを収集し、スキャンする工程に最も多くの工数が必要となった。

表 7.5　データ構造化処理における作業種別

作業種別	説明
紙スキャンが必要	紙資料をスキャナーでスキャンする作業を実施する必要があり、その後 Veda で構造化処理、クレンジング等を行う
Excel などのクレンジングが必要	Excel などの電子データがすでにあり、スキャン作業がほとんど不要であり、Veda で Excel データの構造化処理、クレンジング等を行う

そのまま csv 等が使える	CSV などの構造化データがすでにあり、データ構造化が不要

表 7.6　作業種類別平均人的コスト

作業種類	紙スキャン	構造化処理（人日）	クレンジング処理（人日）
紙スキャンが必要	25 人日	2 人日	3 人日
Excel などのクレンジングが必要	-	1.5 人日	2 人日
そのまま csv 等が使える	-	-	0.3 人日

また、ページ数が多いなどで最も作業時間がかかった帳票の例を以下に示す。この例では、特定市域に限定したスキャンだけでも 9000 件存在し、なおかつスキャン対象のページ数が多かったため多くの人月工数を要した。仮に全国規模で作業を行った際の、最大作業規模は以下の通りである。

表 7.7　紙スキャン作業の見積もり人的コスト（今年度最大規模ケース）

対象地域範囲	スキャン件数	工数（人月）	備考
特定地域	9000 件	3 人月	今年度実績
全国	42000 件	14 人月	今年度実績より算出

7.7　本章のまとめ

本章で示したとおり、コスト総計¥28,340,656 のうち LLM API 利用料（AWS Bedrock）が約 80%を占める偏重構造である。次年度はモデルの使い分け・プロンプトキャッシング・不要インスタンスの停止等の最適化施策を通じて、精度を維持しつつコスト効率を改善することが課題となる。

第 8 章　成果と課題

本章では、Veda の 2025 年度実証を通じて得られた成果と、実証を進める中で明らかになった課題・対応策を総括する。各章で記述した技術検証の結果を踏まえ、前年度（2024 年度）との比較も交えながら整理する。

8.1　本実証で得られた成果

2025 年度の実証では、2024 年度の課題を起点に取り組んだ 3 つの領域でそれぞれ大きな成果を得た。

8.1.1　LINKS Veda のシステムアーキテクチャ刷新と UI/UX 改善

2025 年度の Veda では、データパイプライン設計と UI/UX の両面で大幅な刷新を行った。

データアーキテクチャとしては、メダリオンアーキテクチャ（Bronze/Silver/Gold 層構成）を採用し、生データ・クレンジング済みデータ・分析用データを段階的に管理できる設計に移行した。これにより各層のデータを独立して検証・再処理できるようになり、AI 処理を含むデータパイプラインの品質管理が現実的な水準に達した。また、Silver 層を PostgreSQL で管理し Gold 層を Parquet ファイルとして GCS 上に出力する構成により、スケーラブルかつ安定したデータ出力基盤が整備された（詳細は第 3 章参照）。

UI/UX の面では、2024 年度に課題であった「ワークフロー概念の難解さ」を根本から解決するため、データ構造化のフローをフォーム形式のステップに再設計した。スキーマサジェッション（AI 自動提案）から構造化処理・結果確認までを順を追って進められるようになり、初めて利用するユーザーでも説明なしに一連の処理を完了できる体験を実現した。また、構造化結果と元の PDF を左右に対比表示し、各項目の抽出元領域をバウンディングボックスでハイライトする機能により、AI の出力根拠を直感的に確認できるようになった（詳細は第 3 章参照）。

8.1.2　データ構造化技術の精度・スケーラビリティ向上

2024 年度時点では数十件の PDF を同時に処理することが実質的な上限であったが、2025 年度は Cloud Run の並列処理アーキテクチャ（1 コンテナ 1 ファイル設計）と AWS Bedrock の API レートリミット引き上げにより、大量のファイルを並列的に構造化処理できるようになった。処理速度も向上しており、大量データを実運用レベルで扱える基盤が整いつつある。

精度の面では、OCR＋LLM 構成（Azure Document Intelligence + Claude Sonnet）が実証を通じて最も安定した結果を示すことを確認し、これを標準構成として確立した。打ち消し線検知と分岐処理の二段判定アーキテクチャにより、手書き訂正が含まれる帳票でも精度を維持できる仕組みを実装した。プロンプトエンジニアリング（特に Few-shot）を全ユースケースに標準導入し、選択式項目の精度改善も実現した（詳細は第 4 章参照）。

8.1.3　汎用的 BI ツールの構築と AI チャットによる分析体験の実現

2024 年度はユースケースごとに個別のフロントエンドアプリケーションを開発しており、新たなユースケースに対応するたびに一からアプリを構築する運用となっていた。2025 年度はこれを廃止し、すべてのユースケースに対応する汎用的な BI を構築した。

LINKS BI では自然言語で AI に指示するだけで、DuckDB SQL によるデータ集計・グラフ可視化・地図表示・ダッシュボード作成を一貫して行うことができるようになった。SQL や Python の知識を持たない国交省職員でも、チャットでやり取りするだけで探索的なデータ分析を実行できる体験が実現した。さらに国会答弁案生成機能のように行政業務に特化した AI 機能を追加できるプラグインアーキテクチャを採用しており、ユースケースの拡張性も確保されている（詳細は第 5 章参照）。

8.2　明らかになった課題と対応方針

実証を進める中で、技術面・運用面の双方において今後対処すべき課題が明らかになった。以下に主要な課題と対応方針を整理する。

8.2.1　Veda のデータ加工機能（テーブルアクション）の使い勝手

データ構造化機能は大幅に改善したが、構造化後のデータを分析可能な状態に整備するテーブルアクション機能には、Excel と比較した際の操作性・再利用性に課題が残る（6.2.1 節・6.2.3 節参照）。次年度は、複数のテーブルアクションをパイプラインとして保存・再利用できる機能や、データ加工の中間状態をステップごとに確認できる機能の実装を優先課題として取り組む予定である。

8.2.2　データ構造化精度のさらなる改善

現在の構造化精度は 2024 年度比で大きく向上しているが、それでも一定の割合で正確に抽出できない項目が残る。特に精度が低い傾向があるのは、打ち消し線を踏まえた情報抽出や判読の難しい手書き文字、略語で記載されている特殊な表記揺れといったケースなどである。

また、現状は全ユースケースに共通の汎用プロンプトで LLM を呼び出しており、各様式・各業務特有のデータ特性に合わせたコンテキストを LLM に渡せていない。それぞれのユースケースが持つ書類の特性（記入ルール・用語・様式の構造等）をプロンプトに反映すると、LLM の能力をより引き出せると考えられる。次年度はプロンプトの自動最適化フレームワークの導入を検討しており、ユースケースごとに最適化されたプロンプトを動的に生成・適用する仕組みを構築し、構造化精度のさらなる向上を目指す。

8.2.3　BI の生成結果の再現性

AI チャットの生成結果の再現性に課題があり、同一のプロンプトとデータに対して試行ごとに異なる結果が生成されるケースがある（6.3.1 節参照）。根本原因はユーザーのプロンプトに残る曖昧性にあり、AI が前提条件を確率的に解釈することで出力にブレが生じる。対応策とし

て、AI がユーザーに事前確認を行い曖昧さを解消してから処理を実行するプランニングモードの導入を検討している（5.9.4 項参照）。

8.3　次年度（2026 年度）に向けた展望

2025 年度までは Veda を実証実験として構築・運用してきたが、2026 年度はプロダクションレベルの品質へ引き上げることが求められる。この移行にあたっては、新機能の開発だけでなく、実際に組織的な運用を支えるための整備が必要になる。

具体的には以下の取り組みが 2026 年度の課題として挙げられる。

- オンボーディング支援機能の充実：第 6 章の実証評価で操作習熟に一定の学習コストがかかることが課題として確認された（6.3.1 節参照）。初めて利用するユーザーがシステムの使い方を自己習得できるよう、ガイドや操作説明をシステム内に組み込む。
- 利用状況の監視強化：第 6 章で定量的な利用データの整理が今後の課題として挙がっている。どの機能がどの程度使われているか、エラーや不満が発生している箇所はどこかを把握できる観測基盤を整備し、継続的な改善サイクルを確立する。
- 運用業務フローの策定：Veda はまだ試験運用段階であり、国交省各部門へのユーザーアカウントの発行・管理フローなど本格運用必要な様々な運用フローが整備しきれていない。アカウント管理方針、障害発生時の対応手順書、データ追加・ワークフロー設定のサポートフロー等、実運用を継続的に回していくための業務手順を整備する。
- システムの安定性・セキュリティの強化：第 7 章のコスト分析で示したインフラ構成を前提に、プロダクション環境として求められるサービスレベル（可用性・応答性・データ保護）の基準を定め、それを満たすための設計・監視体制を確立する。

これらの取り組みを通じて、Veda を研究開発・実証の成果物から、実際に行政業務で継続的に活用される社会基盤へと発展させることが 2026 年度の中心的な目標となる。

本レポートでは、LINKS Veda の技術選定の判断根拠・検証プロセス・実証結果を記録した。本レポートが、次年度以降の開発・運用における意思決定の基盤として活用されることを期待する。

第 9 章　用語集

本章では、本ドキュメントで使用するプロジェクト固有の用語・業務ドメイン用語・アプリケーション内の概念用語を一覧で定義する。一般的な IT 用語・ライブラリ名・技術用語については各章の解説を参照すること。

9.1　プロジェクト・システム用語

本実証に関わるプロジェクト名称・組織名称・システム名称を以下に定義する。

表 9.1　プロジェクト・システム用語

用語	定義
LINKS Veda	Project LINKS におけるデータ構築・活用システムの総称。「データ構築モジュール」と「データ活用モジュール（LINKS BI）」の 2 つのサブシステムで構成される。非構造データの自動構造化とその分析・可視化を一体的に提供するシステム。略して Veda。
データ構築モジュール	Veda におけるサブシステムの一つで、PDF 等の非構造データを AI を利用して構造化する。
データ活用モジュール（LINKS BI）	Veda を構成するデータ活用モジュール。データ構築モジュールが構造化したデータを AI チャットおよびダッシュボードを通じてインタラクティブに分析・可視化するサブシステム。
ユースケース（UC）	本実証において検証対象とした行政業務の具体的な適用事例の単位。

9.2　業務ドメイン用語

本実証の対象となる行政業務・データ処理に関連する用語を以下に定義する。

表 9.2　業務ドメイン用語

用語	定義
EBPM	Evidence-Based Policy Making（エビデンスに基づく政策立案）の略。データや統計的エビデンスを根拠として政策を設計・評価する考え方。本プロジェクトの目的の一つとして位置づけられている。
行政文書	国土交通省をはじめとする行政機関が作成・収集する公文書の総称。帳票・申請書類・報告書類・答弁書など、多様な形式が存在する。
様式	帳票の書式・レイアウトのこと。同一種類の帳票でも複数の様式が存在することがあり（例：1 号様式・2 号様式）、様式ごとに異なるスキーマが必要となる場合がある。

9.3　データ構築モジュール用語

データ構築モジュール内で使用される概念・機能名称を以下に定義する。

表 9.3　データ構築モジュール用語

用語	定義
アセット	アップロードされた元ファイル（PDF・Excel・Word 等）の管理単位。Bronze 層に格納され、構造化処理の入力として参照される。
ワークフロー	LINKS Veda におけるデータ構造化処理の一連の設定と処理手順の単位。スキーマ定義・処理オプション・実行結果を一体として管理する。
スキーマ	データ構造化処理において抽出する項目の定義。どの情報を・どのデータ型で・どのキー名で抽出するかを記述する。
スキーマサジェッション	アップロードされた資料を AI が自動解析し、構造化に適したスキーマ候補を提案する機能。ユーザーがゼロからスキーマを定義する手間を削減する。
データ構造化処理	アセット（PDF 等）からスキーマに基づいてデータを抽出し、テーブル形式の構造化データとして出力する処理。OCR＋LLM を中心に実装されている。
データテーブル	構造化結果、または構造化済データを CSV 等の形式でインポートしたテーブル形式のデータ。BI での利用や、テーブルアクションの実行対象となる。
テーブルアクション	データテーブルに対して UI から実行できるデータ変換・加工操作の総称。型変換・正規化・和暦変換・住所正規化・テーブル結合など、行政データに特化した多数の処理を提供する。
バウンディングボックス（bbox）	構造化処理において、各フィールドの値が元資料のどの領域から抽出されたかを示す矩形座標情報。プレビュー画面で視覚的に確認でき、結果の検証に活用する。
信頼度スコア	LLM による抽出結果に対してシステムが付与するスコア。抽出が失敗している可能性が高いフィールドや不確実性が高いフィールドを示し、ユーザーによる確認優先度の判断に使用する。

メダリオンアーキテクチャ	データを品質レベルに応じて Bronze・Silver・Gold の 3 層に分けて管理するデータパイプライン設計パターン。データ構築モジュールにおけるデータ管理の根幹として採用している。
Bronze 層	メダリオンアーキテクチャの第 1 層。ユーザーがアップロードした元データ（アセット）をそのまま保管する生データ層。
Silver 層	メダリオンアーキテクチャの第 2 層。Bronze 層のデータを AI で構造化・クレンジングした中間データを管理する層。
Gold 層	メダリオンアーキテクチャの第 3 層。Silver 層のデータを分析最適化し、管理する層。LINKS BI のデータソースとなる。
DuckDB	オープンソースのインプロセス（別途サーバーを立てることなく、アプリケーション本体と同じプロセス内で動作する方式）のデータベース。列指向の特性から集計・フィルタリング処理を高速に実行できる。データ構築モジュールではサーバーサイドでテーブルアクションの実行に、LINKS BI では WebAssembly 版（DuckDB-Wasm）としてブラウザ内データ分析に使用する。
Parquet	Apache Software Foundation が管理するオープンソースの列指向ファイルフォーマット。高い圧縮効率と分析クエリに対する I/O 効率を特徴とする。本システムでは Gold 層のデータ格納形式として採用している。

9.4　データ活用モジュール（LINKS BI）アプリケーション概念

LINKS BI のアプリケーション内で使用される概念・機能名称を以下に定義する。

表 9.4　LINKS BI アプリケーション概念

用語	定義
AI チャット機能	LINKS BI の中核機能。ユーザーが自然言語でデータ分析の指示を入力すると、AI が SQL クエリを生成・実行し、グラフや地図で結果を表示する機能。
ダッシュボード	LINKS BI において、AI チャットで生成したグラフや地図を自由にレイアウトして 1 画面にまとめる機能。複数のウィジェットをグリッドレイアウト上に配置する。
ウィジェット	ダッシュボード上に配置される個別の可視化コンポーネント（グラフ・地図・テーブル等）の単位。

スキル	LINKS BI の AI チャットにおいて、特定のデータ分析タスクを実行するための専門的なプロンプトとツール設定のセット。国会答弁案生成・経路探索など業務別に定義される。
Function calling（ツール呼び出し）	AI がユーザーの指示に応じて、SQL クエリ実行・グラフ生成・地図表示などの特定ツールを自動的に選択・実行する仕組み。LINKS BI の AI チャットの根幹技術。
AI 回帰テスト	LINKS BI の AI チャット機能について、特定のシナリオに対する応答が期待通りであるかを繰り返し検証するテスト手法。スキルチューニングによる改善効果の測定にも使用する。
スキルチューニング	AI チャットの再現性・精度向上を目的として、スキルファイル（プロンプト・ルール・ガイドライン）を修正・最適化する作業。

国土交通分野のデータ整備・活用・オープンデータ化の推進に向けた
データ構築基盤の技術実証調査
技術検証レポート

2026 年 3 月 発行

委託者：国土交通省 総合政策局 情報政策課