実験ノート検索システム - 機能拡張版 仕様書

プロジェクト概要

目的

LangChain v2で構築した実験ノート検索システムをベースに、ユーザビリティとカスタマイズ性を大幅に向上させた機能拡張版を開発する。

ベースシステム

- **元システム**: `/Users/nori8774/LangChain_v2`
- **コア技術**: LangChain + LangGraph + ChromaDB + Cohere Reranking
- **既存機能**: ベクトル検索、多角的クエリ生成、材料名正規化、比較分析レポート生成

最終ゴール

Vercelにデプロイし、研究室・チーム内で共有できる実験ノート検索Webアプリケーション

技術スタック

フロントエンド

- **フレームワーク**: Next.js 15 (App Router)
- **言語**: TypeScript
- **UI**: React 19 + Tailwind CSS
- **状態管理**: React Context または Zustand
- **マークダウン表示**: React Markdown
- **デザインシステム**: 既存UIのカラースキーム・トーンを継承
  - プライマリカラー: `#1c3c3c` (深緑系)
  - ユーザーメッセージ: `#076699` (青系)
  - 背景: `#f9f9f9` (ライトグレー)
  - 成功: `#10b981`, 警告: `#f59e0b`, エラー: `#ef4444`
  - フォント: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto

バックエンド

- **API**: FastAPI (Python 3.12+)
- **フレームワーク**: LangChain + LangGraph
- **ベクトルDB**: ChromaDB
- **Embedding**: OpenAI Embeddings (ユーザー選択可能)
- **Reranking**: Cohere Rerank API

インフラ・デプロイ ⭐

- **フロントエンド**: Vercel
- **バックエンド**: Vercel Serverless Functions または Railway / Render

- **ファイルストレージ**: ローカルファイルシステム
  - ユーザー指定フォルダ（新規ノート、アーカイブ）
  - 設定画面でパス設定可能
- **ベクトルDB**: ローカルChromatDB
  - ユーザー指定フォルダ（デフォルト: `./chroma_db`）
  - 永続化ストレージ（ディスク保存）
- **注意**: バックエンドをVercel Serverless Functionsで動かす場合、ファイル永続化に制限あり → Railway/Renderでの常時起動サーバー推奨

認証・セキュリティ

- **認証**: 不要（URLアクセスベース）
- **APIキー管理**: ユーザーがブラウザ上で入力・保存（localStorage）
  - OpenAI API Key
  - Cohere API Key
  - Google Drive API Key (optional)

新規追加機能（詳細仕様）

1. プロンプト管理画面

概要

現在コード内にハードコードされているプロンプトをUI上で編集・管理できるようにする。

機能要件

- **プロンプト一覧表示**
  - 正規化ノード用プロンプト
  - クエリ生成ノード用プロンプト（ベテラン/新人/マネージャー視点）
  - 比較ノード用プロンプト
- **編集機能**
  - テキストエリアで直接編集
  - プレビュー機能（変数の展開イメージ表示）
  - **デフォルト値へのリセット機能** ⭐
    - 各プロンプト単位でリセット可能
    - 全プロンプト一括リセット機能
    - リセット前に確認ダイアログ表示
    - デフォルトプロンプトは `agent.py` から取得
- **保存・適用**
  - ブラウザのlocalStorageに保存
  - リアルタイム適用（次回検索から反映）
- **テンプレート変数**

    - `{input_purpose}`, `{normalized_materials}`, `{input_methods}`, `{instruction}` などの変数を使用可能
    - 変数のドキュメント表示

UI設計

- 設定画面（サイドバーまたは専用ページ）
- プロンプト種別ごとのタブまたはアコーディオン
- Markdownプレビュー機能

---

2. 検索結果コピー機能

概要

検索結果（上位3件）の材料・方法をワンクリックで検索条件入力欄にコピーできる。

機能要件

- **コピーボタンの配置**
    - 各実験ノート（3件）の「材料」「方法」セクションにコピーボタン
    - クリックで検索条件入力欄に自動反映
- **コピー動作**
    - 「材料」をコピー → 材料入力欄に上書き or 追記（選択可能）
    - 「方法」をコピー → 方法入力欄に上書き or 追記（選択可能）
    - コピー後に通知（トースト表示）
- **フォーマット保持**
    - 箇条書き、改行などのMarkdown形式を維持してコピー

UI設計

- 各ノートの材料・方法セクションの右上にアイコンボタン
- ホバーで「材料をコピー」「方法をコピー」のツールチップ表示

---

3. 検索履歴管理

概要

検索クエリごとに結果を保存し、別タブで履歴を閲覧・再利用できる。

機能要件

- **検索履歴の保存**
    - 検索クエリ（目的・材料・方法・重点指示）

    - 検索日時
    - リランキング後の上位10件の実験ノートID + スコア
    - 保存先: localStorage or サーバーDB

- **履歴一覧表示**
    - 専用タブまたはページ
    - 検索日時、クエリの要約表示
    - クリックで詳細表示

- **詳細表示**
    - 上位10件の実験ノートIDをランキング形式で表示
    - 各IDをクリック → ノートビューワーで全文表示（機能4と連携）

- **再検索機能**
    - 過去のクエリを検索条件欄に復元して再検索

データ構造

```
interface SearchHistory {
  id: string;
  timestamp: Date;
  query: {
    purpose: string;
    materials: string;
    methods: string;
    instruction?: string;
  };
  results: {
    noteId: string;
    score: number;
    rank: number;
  }[];
}
```

UI設計⭐

- タブ: 「検索」「履歴」
- **履歴: テーブル形式で表示**
    - カラム: 日時 | 検索クエリ（要約） | 上位10件のノートID
    - ノートIDはクリッカブル（クリックでビューワー表示）
    - ソート機能（日時順、クエリ名順）
    - フィルタリング機能（キーワード検索）
- 詳細表示: ノートIDクリック → モーダルでビューワー表示

---

4. 実験ノート直接閲覧機能

概要

実験ノート番号を入力すると、該当ノートをビューワーで表示。目的・材料・方法ごとにコピーボタンを配置。

機能要件

- **番号入力機能**
  - テキストボックスに実験ノートID（例: ID3-14）を入力
  - 「表示」ボタンまたはEnterキーで取得
- **ノートビューワー**
  - Markdownレンダリング表示
  - セクション自動パース:
    - 目的・背景
    - 材料
    - 方法
    - 結果（あれば）
- **セクション別コピーボタン**
  - 各セクション（目的、材料、方法）の右上にコピーボタン
  - クリックで検索条件入力欄に反映
- **エラーハンドリング**
  - 存在しないIDの場合: エラーメッセージ表示

API設計

```
GET /notes/{note_id}
Response: {
  id: string;
  content: string; // Markdown全文
  sections: {
    purpose?: string;
    materials?: string;
    methods?: string;
    results?: string;
  }
}
```

UI設計

- 専用ページまたはモーダル
- 左側: 入力フォーム、右側: ビューワー（2カラムレイアウト）
- レスポンシブ対応

---

5. モデル選択UI

概要

OpenAIのEmbeddingモデルと検索用LLMをUI上で選択可能にする。

機能要件

- **選択可能なモデル**
  - **Embeddingモデル**:
    - text-embedding-3-small（デフォルト）
    - text-embedding-3-large
    - text-embedding-ada-002
  - **検索用LLM**:
    - gpt-4o-mini（デフォルト）
    - gpt-4o
    - gpt-4-turbo
    - gpt-3.5-turbo
- **設定画面**
  - サイドバーまたは専用設定ページ
  - ドロップダウンで選択
  - 各モデルの説明（コスト、性能、速度）を表示
- **保存と適用**
  - localStorageに保存
  - サーバーへのリクエスト時に選択モデルを送信
  - バックエンドで動的にモデルを切り替え

API変更

- リクエストに `embedding_model` と `llm_model` パラメータを追加
- バックエンドで動的にモデルをインスタンス化

---

6. RAG性能評価機能

概要

事前に設定した正解ランキングと検索結果を比較し、nDCG等の評価指標でRAGの性能を測定する。

機能要件

- **評価用テストセット管理** ⭐
  - **Excel/CSVファイルからインポート**

        - ファイルフォーマット:
            - カラム: `test_case_id`, `query_purpose`, `query_materials`, `query_methods`, `note_id`, `rank`, `relevance`
            - 例: `TC001, ○○の合成, 試薬A..., 手順1..., ID3-14, 1, 5`
        - ドラッグ&ドロップまたはファイル選択でアップロード
        - プレビュー機能（読み込み後に内容確認）
    - テストケースの手動作成・編集
    - 各テストケース:
        - クエリ（目的・材料・方法）
        - 期待される正解ランキング（実験ノートID順）
        - 重み付け（関連度: 1-5等）
    - テストセットのエクスポート（Excel/CSV）

- **評価実行**
    - テストケースを選択して検索実行
    - リランキング結果（上位10件）を取得
    - 評価指標の計算:
        - **nDCG@10** (Normalized Discounted Cumulative Gain)
        - **Precision@K** (K=3, 5, 10)
        - **Recall@K**
        - **MRR** (Mean Reciprocal Rank)

- **結果表示**
    - スコア表示（数値 + グラフ）
    - 正解ランキングと実際のランキングの比較表示
    - 差分の可視化（色分け、順位変動矢印）

- **バッチ評価**
    - 複数テストケースをまとめて評価
    - 平均スコア、最良/最悪ケースの表示

データ構造

```
interface TestCase {
  id: string;
  name: string;
  query: {
    purpose: string;
    materials: string;
    methods: string;
  };
  groundTruth: {
    noteId: string;
    relevance: number; // 1-5
  }[];
}
```

```
interface EvaluationResult {
  testCaseId: string;
  metrics: {
    ndcg_10: number;
    precision_3: number;
    precision_5: number;
    precision_10: number;
    recall_10: number;
    mrr: number;
  };
  ranking: {
    noteId: string;
    rank: number;
    score: number;
    groundTruthRank?: number;
    relevance?: number;
  }[];
}
```

評価指標の計算式

- **nDCG@K**: `DCG@K / IDCG@K`
  - DCG = Σ (relevance_i / log2(i+1)) for i=1 to K
  - IDCG = 理想的なランキングでのDCG

UI設計

- 専用評価ページ
- 左側: テストケース一覧、右側: 評価結果表示
- グラフ: レーダーチャート、比較テーブル

データ管理・ストレージ戦略⭐

実験ノートファイル（.md）

- **保存先**: ローカルファイルシステム
- **管理方法**:
  - **新規ノート用フォルダ**: ユーザーが指定（例: `./notes/new/`）
  - **アーカイブフォルダ**: DB登録済みノート用（例: `./notes/archived/`）
  - **フロー**:
    1. 新規ノートを指定フォルダに配置
    2. DB取り込み実行（後述の新出単語抽出処理含む）
    3. 取り込み後、以下のアクション選択可能:
       - ファイル削除
       - アーカイブフォルダへ移動
       - そのまま残す（何もしない）
    4. 設定画面でデフォルトアクションを設定可能

ベクトルデータベース（ChromaDB）

- **保存先**: ローカルファイルシステム
- **フォルダ指定**: 設定画面でDBフォルダパスを指定可能
  - デフォルト: `./chroma_db`
  - 変更例: `/Users/username/Documents/experiment_db/chroma_db`
- **データ永続化**: ディスク上に保存、再起動後も維持

ユーザー設定・履歴データ

- **保存先**: localStorage（ブラウザ）
- **保存内容**:
  - APIキー（暗号化推奨）
  - カスタムプロンプト
  - モデル選択設定
  - 検索履歴
  - 評価用テストケース

---

7. 新出単語抽出と正規化辞書管理 ⭐

概要

新規ノート取り込み時に、未知の化学物質名を自動抽出し、表記揺れか新規物質かを判定して正規化辞書を更新する。

機能要件

- **新出単語の自動抽出**
  - 新規ノートの材料セクションから単語を抽出
  - 既存の正規化辞書と照合
  - 辞書にない単語をリストアップ
- **表記揺れ判定支援**
  - LLMを使用して類似単語を検索
  - 候補リスト表示:
    - 既存辞書内の類似単語（編集距離、意味的類似度）
    - 判定: 「既存の表記揺れ」or「新規物質」
  - ユーザーが判定を確認・修正
- **正規化辞書の更新**
  - 表記揺れの場合: 既存エントリに追加
  - 新規物質の場合: 新規エントリ作成
  - 更新内容のプレビュー表示
  - 一括更新またはレビュー後に個別更新
- **辞書管理UI**

- 正規化辞書の閲覧・検索
- エントリの編集・削除
- 辞書のエクスポート/インポート（YAML/JSON/CSV）
- 更新履歴の表示

データ構造

```
interface NormalizationEntry {
  canonical: string; // 正規化後の名称
  variants: string[]; // 表記揺れのリスト
  category?: string; // カテゴリ（試薬、溶媒、etc）
  note?: string; // メモ
  created_at: Date;
  updated_at: Date;
}

interface NewTermSuggestion {
  term: string; // 新出単語
  source_note_id: string; // 出現したノートID
  similar_terms: { // 類似単語候補
    term: string;
    similarity: number;
    canonical: string;
  }[];
  user_decision: 'new' | 'variant' | 'skip';
  canonical?: string; // 紐付け先（variantの場合）
}
```

ワークフロー

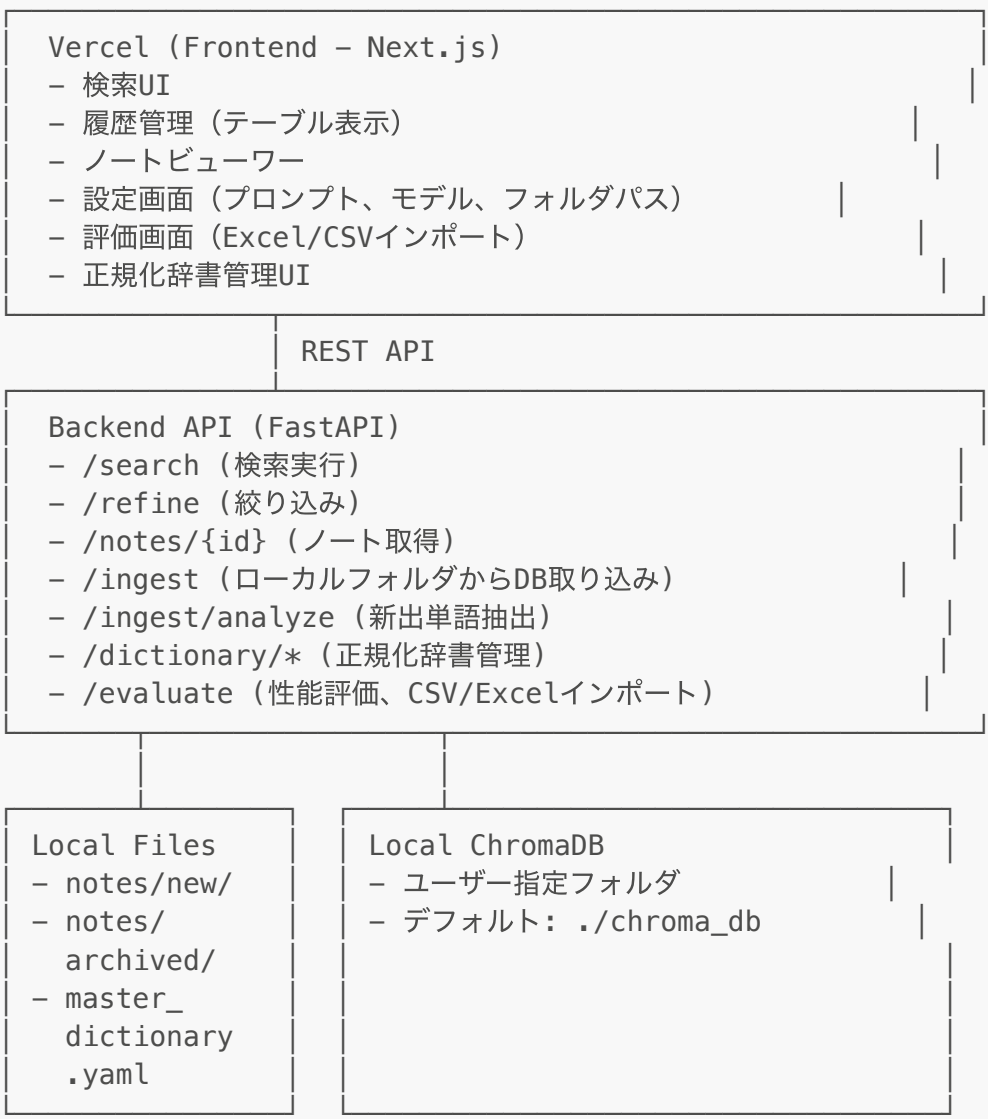
1. 新規ノート取り込み実行
2. 材料セクションから単語抽出
3. 既存辞書と照合、新出単語をリストアップ
4. LLMで類似単語検索（類似度計算）
5. UI上でユーザーに判定を促す
6. ユーザーが判定（表記揺れ/新規物質/スキップ）
7. 正規化辞書を更新
8. ノートをベクトル化してDBに追加
9. ファイル処理（削除/移動/保持）

UI設計

- ノート取り込み時に「新出単語検出」モーダル表示
- テーブル表示: 新出単語 | 類似候補 | 判定 | アクション
- 判定: ドロップダウン（既存の〇〇 / 新規物質 / スキップ）
- 一括承認ボタン、個別レビューボタン

システムアーキテクチャ

全体構成図 ⭐



API設計（追加エンドポイント）

1. ノート取得

```
GET /notes/{note_id}
Response: {
  success: boolean;
  note: {
    id: string;
    content: string;
    sections: {
      purpose?: string;
      materials?: string;
      methods?: string;
      results?: string;
```

```
    };
  }
}
```

2. ノート取り込み（増分更新）⭐

```
POST /ingest
Request: {
  source_folder: string; // 新規ノートフォルダパス
  post_action: 'delete' | 'archive' | 'keep'; // 取り込み後のアクション
  archive_folder?: string; // アーカイブ先（actionがarchiveの場合）
}
Response: {
  success: boolean;
  new_notes: string[]; // 取り込んだノートIDリスト
  skipped_notes: string[]; // 既存のためスキップしたID
  new_terms: { // 新出単語リスト
    term: string;
    note_id: string;
    similar_candidates: { term: string; similarity: number; canonical:
string }[];
  }[];
}
```

3. 新出単語分析

```
POST /ingest/analyze
Request: {
  note_ids: string[]; // 分析対象ノートID
}
Response: {
  new_terms: {
    term: string;
    note_id: string;
    similar_candidates: {
      term: string;
      similarity: number;
      canonical: string;
    }[];
  }[];
}
```

4. 正規化辞書更新

```
POST /dictionary/update
Request: {
```

```
  updates: {
    term: string;
    decision: 'new' | 'variant';
    canonical?: string; // variantの場合、紐付け先
    category?: string;
    note?: string;
  }[];
}
Response: {
  success: boolean;
  updated_entries: number;
}

GET /dictionary
Response: {
  entries: {
    canonical: string;
    variants: string[];
    category?: string;
    note?: string;
  }[];
}
```

5. 性能評価（Excel/CSVインポート対応） ⭐

```
POST /evaluate/import
Request: multipart/form-data
  file: Excel/CSVファイル
Response: {
  success: boolean;
  test_cases: {
    id: string;
    query: { purpose, materials, methods };
    ground_truth: { note_id, rank, relevance }[];
  }[];
}

POST /evaluate
Request: {
  test_case_id: string;
  query: { purpose, materials, methods };
  ground_truth: { note_id, rank, relevance }[];
}
Response: {
  success: boolean;
  metrics: {
    ndcg_10: number;
    precision_k: { k3: number; k5: number; k10: number };
    recall_10: number;
    mrr: number;
  };
```

```
  ranking: { note_id, rank, score }[];
  comparison: { // 正解 vs 実際のランキング比較
    note_id: string;
    expected_rank: number;
    actual_rank: number;
    relevance: number;
  }[];
}
```

4. モデル設定付き検索

```
POST /search
Request: {
  purpose: string;
  materials: string;
  methods: string;
  embedding_model?: string; // デフォルト: text-embedding-3-small
  llm_model?: string; // デフォルト: gpt-4o-mini
  custom_prompts?: {
    normalize?: string;
    query_generation?: string;
    compare?: string;
  }
}
```

UI/UX設計

ページ構成 ⭐

1. **検索ページ（メイン）**
   - 検索条件入力フォーム
   - 検索結果表示（上位3件 + セクション別コピーボタン）
   - 絞り込み機能
2. **履歴ページ**
   - 検索履歴一覧（テーブル表示：日時 | クエリ要約 | 上位10件ノートID）
   - ノートIDクリックでビューワー表示（モーダル）
   - ソート・フィルタリング機能
   - 再検索機能
3. **ノートビューワーページ**
   - 実験ノートID入力
   - Markdown表示（セクション別コピーボタン付き）
   - 目的・材料・方法の各セクションにコピーボタン

4. **ノート管理ページ** ⭐
    - 新規ノート取り込み
    - 新出単語検出・判定UI
    - 取り込み後のアクション設定（削除/移動/保持）
    - 正規化辞書管理（閲覧・編集・エクスポート）
5. **設定ページ**
    - APIキー入力（OpenAI, Cohere）
    - モデル選択（Embedding, LLM）
    - プロンプト管理（編集 + 初期設定リセット）
    - フォルダパス設定:
        - 新規ノートフォルダ
        - アーカイブフォルダ
        - ChromaDBフォルダ
    - デフォルトアクション設定
6. **評価ページ**
    - テストケース管理（Excel/CSVインポート + 手動作成）
    - 評価実行（単体 or バッチ）
    - 結果表示（グラフ + 比較テーブル）
    - nDCG, Precision, Recall, MRR表示

レイアウト

- **ヘッダー**: ナビゲーションメニュー（検索 / 履歴 / ビューワー / 評価 / 設定）
- **サイドバー**: 設定クイックアクセス、APIキーステータス
- **メインコンテンツ**: ページごとのコンテンツ
- **フッター**: バージョン情報、ヘルプリンク

開発フェーズ・優先順位 ⭐

Phase 1: 基盤整備（Week 1-2）

- [ ] プロジェクトセットアップ（Next.js + FastAPI）
- [ ] 既存UIのカラースキーム・デザインシステムの整理
- [ ] 既存コードの移行・リファクタリング
- [ ] ローカルストレージ管理機能（フォルダパス設定）
- [ ] API設計とエンドポイント実装
- [ ] 基本的なUI構築

Phase 2: コア機能実装（Week 3-4）

- [ ] 機能1: プロンプト管理画面（初期設定リセット機能含む）
- [ ] 機能2: 検索結果コピー機能
- [ ] 機能4: ノートビューワー（セクション別コピーボタン）
- [ ] 機能5: モデル選択UI

- ☐ 増分DB更新（既存ノートスキップ機能）

Phase 3: ノート管理・辞書機能実装（Week 5-6）

- ☐ 機能7: 新出単語抽出と正規化辞書管理
- ☐ 取り込み後のファイルアクション（削除/移動/保持）
- ☐ 正規化辞書管理UI
- ☐ LLMによる表記揺れ判定機能

Phase 4: 履歴・評価機能実装（Week 7-8）

- ☐ 機能3: 検索履歴管理（テーブル表示、ノートIDクリック表示）
- ☐ 機能6: RAG性能評価機能
- ☐ Excel/CSVインポート機能
- ☐ nDCG等の評価指標実装
- ☐ バッチ評価機能

Phase 5: UI/UX改善・テスト（Week 9-10）

- ☐ 既存UIデザインとの統一性確認
- ☐ レスポンシブ対応
- ☐ エラーハンドリング・バリデーション
- ☐ パフォーマンス最適化
- ☐ 統合テスト

Phase 6: デプロイ・ドキュメント（Week 11-12）

- ☐ Vercelデプロイ設定
- ☐ バックエンドデプロイ（Vercel Serverless or Railway）
- ☐ 本番環境テスト
- ☐ ユーザーマニュアル作成
- ☐ チーム内説明会・フィードバック収集

セキュリティとプライバシー

APIキー管理

- **保存先**: ブラウザのlocalStorage
- **暗号化**: Web Crypto APIで暗号化（推奨）
- **送信**: HTTPS通信のみ
- **サーバー側**: APIキーは保存せず、リクエストごとに受け取る
- **対象**: OpenAI API Key, Cohere API Key

ファイルアクセス制御

- **ローカルファイル**: バックエンドサーバーが動作するマシン上のファイルシステムにアクセス
- **フォルダ権限**: 指定フォルダへの読み書き権限が必要
- **セキュリティ**: サーバー側でパストラバーサル攻撃対策を実装

CORS設定

- フロントエンド（Vercel）のドメインのみ許可
- ローカル開発環境（localhost:3000）も許可

パフォーマンス最適化

フロントエンド

- Next.js App RouterのServer Componentsを活用
- 検索結果のキャッシング（SWR or React Query）
- 画像の最適化（実験ノート内の画像がある場合）
- コード分割とLazy Loading

バックエンド

- ChromaDBのインデックス最適化
- 検索結果のキャッシュ（Redis）
- 非同期処理（複数クエリ生成の並列実行）

技術的課題と対応策 ⭐

課題1: ローカルファイルシステムへのアクセス

- **課題**: Vercel Serverless Functionsでは永続的なファイルストレージが制限される
- **対応**: バックエンドをRailway/Renderの常時起動サーバーにデプロイ
- **代替**: ボリュームマウントが可能なコンテナベースのデプロイ

課題2: ChromaDBの永続化

- **課題**: サーバー再起動時のDB喪失リスク
- **対応**: ディスクボリュームへの永続化設定、定期バックアップ機能実装
- **監視**: DB状態のヘルスチェック機能

課題3: 新出単語抽出の精度

- **課題**: LLMによる類似単語検索が不正確な場合がある
- **対応**:
  - 編集距離（Levenshtein距離）と意味的類似度の併用
  - ユーザーの判定をフィードバックとして学習
  - 手動での辞書編集機能を提供

課題4: APIコスト管理

- **対応**: ユーザー自身のAPIキーを使用（コスト分散）
- **監視**: リクエスト数・トークン数の表示機能
- **最適化**: キャッシング、バッチ処理、モデル選択による調整

課題5: 大量ノートの検索パフォーマンス

- **対応**:
  - ChromaDBのインデックス最適化
  - ページネーション、フィルタリング機能
  - バックグラウンド処理（新出単語抽出等）

課題6: 複数ユーザー間のデータ分離

- **課題**: 現状は単一サーバー・単一DBのため、チーム内で同じデータを共有
- **対応**:
  - 短期: チーム全体で共有（現行仕様）
  - 長期: ユーザーごとのワークスペース機能を検討

成功指標（KPI）⭐

1. **機能完成度**: 7つの新機能がすべて実装され、動作する
   - プロンプト管理（リセット機能含む）
   - 検索結果コピー
   - 検索履歴管理（テーブル表示）
   - ノートビューワー（セクション別コピー）
   - モデル選択UI
   - RAG性能評価（Excel/CSVインポート）
   - 新出単語抽出・正規化辞書管理
2. **デプロイ成功**: Vercelで安定稼働、チーム内でアクセス可能
   - フロントエンド: Vercel
   - バックエンド: Railway/Render等で永続化対応
3. **検索精度向上**: nDCG@10で0.8以上を達成（評価機能で測定）
   - 正規化辞書の継続的改善により精度向上
4. **ユーザビリティ**: チームメンバーのフィードバックで「使いやすい」評価
   - 既存UIデザインとの統一感
   - コピー機能による作業効率化
   - 履歴機能による再検索の容易さ
5. **パフォーマンス**:
   - 検索レスポンス時間 < 5秒
   - 新出単語抽出処理 < 10秒/ノート
   - 増分DB更新により既存ノートはスキップ
6. **運用効率**:
   - ノート取り込み作業の自動化率 > 80%
   - 正規化辞書の更新頻度（月1回程度）
   - チーム内での利用率 > 70%

参考資料

技術ドキュメント

- LangChain Documentation
- ChromaDB Documentation
- Cohere Rerank API
- Next.js App Router
- Google Drive API

評価指標

- nDCG (Normalized Discounted Cumulative Gain)
- Information Retrieval Metrics

付録: agent.pyの仕様について

重要な注意事項: 既存の `agent.py` に実装されているプロンプト、リランキングアルゴリズム、ワークフローは既に確立されており、**変更しないこと**。

新機能では、これらのプロンプトを**UI上で上書き・カスタマイズ可能**にするが、デフォルト値として既存のプロンプトを保持する。

agent.pyのワークフロー（変更なし）

1. **正規化ノード**: 材料名の表記揺れを統一
2. **クエリ生成ノード**: 多角的な検索クエリを生成（ベテラン/新人/マネージャー）
3. **検索ノード**: ベクトル検索 + Cohereリランキング
4. **比較ノード**: 詳細な比較分析レポートを生成

まとめ ⭐

本仕様書に基づき、既存の実験ノート検索システムをユーザビリティとカスタマイズ性に優れた機能拡張版へと進化させる。

主な改善点

1. **UIのカスタマイズ性向上**: プロンプト管理、モデル選択、既存デザイン継承
2. **ワークフロー最適化**: 検索結果コピー、履歴管理、ノート直接閲覧
3. **インテリジェントなノート管理**: 新出単語抽出、表記揺れ判定、正規化辞書自動更新
4. **評価機能**: Excel/CSVインポート対応、nDCG等の客観的評価指標
5. **柔軟なストレージ**: ローカルベース、フォルダパス指定、増分更新
6. **チーム共有**: Vercelデプロイ、URLベースアクセス、各自のAPIキー使用

デプロイ後の利用イメージ

- 研究室メンバーがURLにアクセス
- 各自のAPIキーを設定画面に入力

- 新規実験ノートを指定フォルダに配置
- 自動で新出単語を検出・辞書更新
- カスタマイズしたプロンプトで高精度検索
- 検索履歴を参照・再利用
- RAG性能を定量的に評価・改善

Vercelでのデプロイにより、研究室・チーム内で容易にアクセス・共有できる環境を構築する。

作成日: 2025-12-19 **最終更新**: 2025-12-19 **バージョン**: 2.0.0 **ベースシステム**: LangChain_v2 実験ノート検索システム **主要な変更点**:

- 既存UIデザインの継承（カラースキーム、トーン）
- プロンプト初期設定リセット機能
- 増分DB更新（新規ノートのみ処理）
- ノート取り込み後のファイルアクション（削除/移動/保持）
- 新出単語抽出と正規化辞書自動更新
- 検索履歴テーブル表示、ノートIDクリック表示
- RAG評価のExcel/CSVインポート
- ローカルストレージ化、フォルダパス指定機能