

検索の内部構造を読む

技術者向け・アドバンス

Plone × LanceDB 検索デモ

Web版の解説を、図と比較表を含めてまとめた持ち帰り資料です。実際の操作画面と最新の説明はサイトで確認できます。

Web版：技術者向け



<https://d3i4vv7cxswlt1.cloudfront.net/>

2026年10月14日版。開発中のデモのため、機能と画面は今後変わる場合があります。記載の操作は、各機能が管理者により有効化されていることを前提とします。

技術者向け

「お城めぐり」で城の記事が見つかる。「電車」で、ファイル名しか付いていない写真が出てくる。この体験の裏には、特徴を数値にするモデル、異なる順位の統合、CMSの権限を守る仕組みがあります。ここでは検索語が結果になるまでを、設計の理由と一緒にたどります。

初めて読む方は、最初の役割分担と1〜7節へ。内部構造を追いたい方は、その後の「アドバンス」でテーブルのキー、世代、検索要求の経路、Voltoへの接続を読み進められます。8〜10節にはコードの地図・検証・運用上の限界をまとめています。

実装の説明は spec-23-volto-search ブランチの a9262d3a35b1 (2026年10月11日) の資料・コードに基づきます。稼働サイトの全設定やデプロイ版が一致することを示すものではありません。図と数値例は、特記しない限り仕組みを説明するための例です。

まず見分ける：Plone標準と、このアドオンの追加機能

ページを作る、写真を保存する、公開前の記事を隠す、APIで取得する。これらの土台はPloneとVoltoが提供します。c2.search.lancedbはその上に、意味と画像の特徴を使う検索経路を加えます。ここを分けて見ると、このパッケージが何を実装し、何を標準機能へ委ねているのかが分かります。



PloneとVoltoの標準機能、c2.search.lancedbが追加する同期・検索・統合処理、LanceDBやE5やSigLIP 2など外部技術の役割を区別する図。

観点	Plone / Voltoが提供する土台	このパッケージが加えること
コンテンツ	ページ・画像・UID・ZODBへの保存	検索用documents・chunks・imagesへの非同期同期

観点	Plone / Voltoが提供する土台	このパッケージが加えること
本文の取り出し	SearchableTextとindexer、ブロックごとの本文抽出	標準の抽出器をworkerのコンテキストで再利用する
検索	ZCatalogによる検索、RESTの@search	LanceDBの全文検索、E5の意味検索、RRFでの順位統合
画像	画像フィールド、保存、@@imagesによる配信	SigLIP 2による内容の特徴計算と、言葉からの画像検索
権限	ロール・ユーザー・公開期間、restrictedな検索	検索側での事前絞り込みと標準の最終認可への接続
画面	Voltoの編集ブロック、RESTの応答形式	条件に合う@searchをLanceDBへ接続し、標準の応答形式へ戻す
タグ	Subjectとしてのタグの保存・編集	画像・文章からの候補判定と、似たコンテンツからのお勧め

このパッケージの価値が見える4つの接続

①別々の単位で探した結果をコンテンツへ揃える。キーワードは文書、ベクトルはチャンク、画像は画像フィールド。UIDへ集約し、RRFで同じ一覧へ統合するところが、検索体験を作ります。

②非同期処理とCMSの権限を両立する。モデルの計算を保存から離しながら、最後の認可は現在のPloneへ戻します。公開・非公開が変わり続けるCMSにとって重要な接続です。

③編集者の作業へつなぐ。画像を内容で探すだけでなく、タグ候補、似た画像、本文からの参照などを通じ、蓄積したコンテンツの発見と整理へ広がります。

④既存の画面やAPIの契約を保つ。検索処理の一部を切り替えても、Voltoが受け取る標準の応答形式や、標準検索を必要とする編集用の用途を保ちます。

E5・SigLIP 2・LanceDBは外部のモデル / ライブラリです。RRFも既存の手法です。ここでいう「追加された新しさ」は、それらをPloneのコンテンツ・権限・更新・画面の契約へ統合した機能と設計にあります。モデルや検索アルゴリズム自体をこのパッケージが発明した、という意味ではありません。

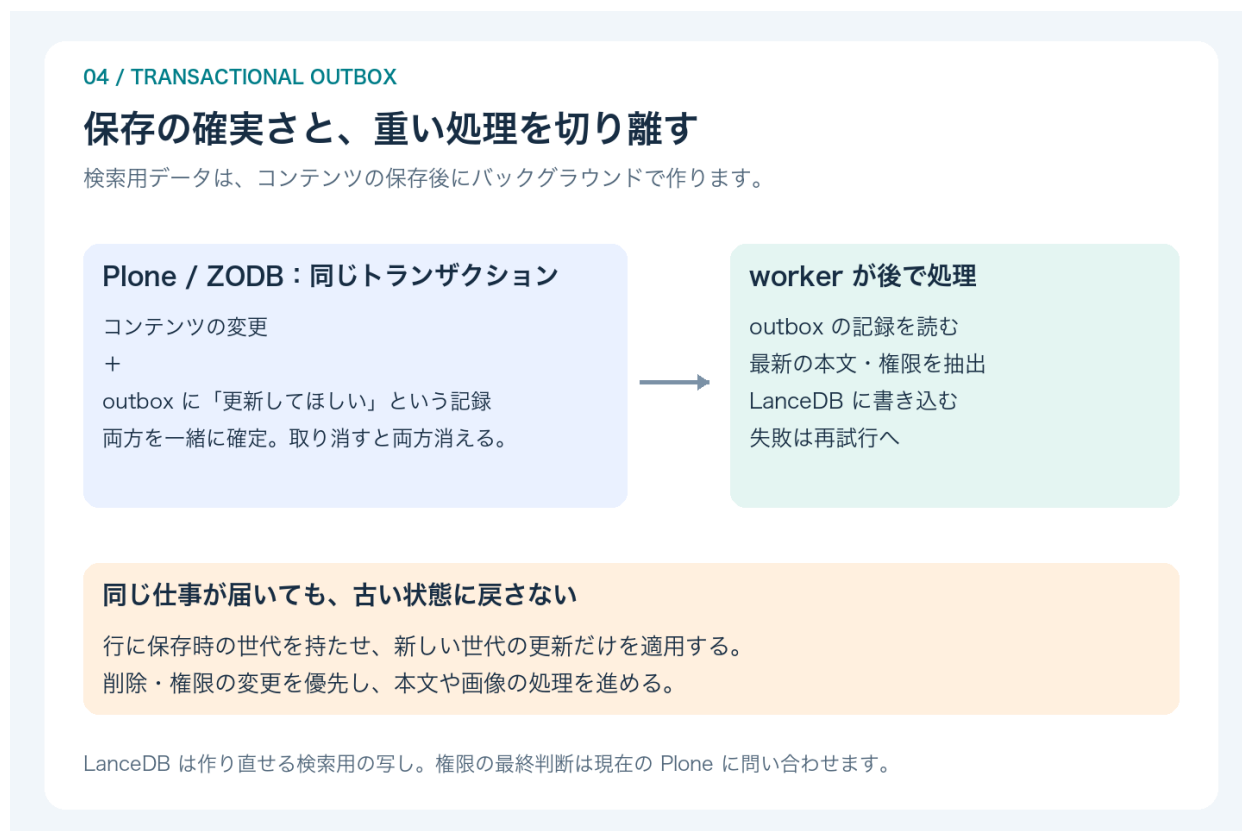
デモサイトの見た目と、検索エンジンの役割

この説明ページの文章・図・表はVoltoの標準ブロックで作っています。/image-searchという画面のURLと、サーバーの@lance-searchというAPIも別のものです。画面をどう配置するかはサイト側の構成、検索候補・順位・認可・応答をどう作るかはバックエンド側の責務として読むと、見た目の機能とパッケージの機能を混同せずに追えます。

実装を読む : [Volto連携の適用範囲](#)

1. 保存と検索をつなぐ、2つの経路

Ploneはページ・画像・公開状態・閲覧権限を管理し、Voltoはその情報をAPIで受け取って画面を表示します。LanceDBには検索に必要な文章、数値の特徴、権限の写しを置きます。Ploneの標準カタログであるZCatalogを正として残すため、検索用データを失っても元のコンテンツから再構築できます。



コンテンツの変更とoutboxへの記録を同じトランザクションで保存し、workerが後からLanceDBを更新します。古い世代の更新は適用しません。

保存ボタンで、モデルの計算を待たせない

文章や写真の特徴を計算する処理は、普通の保存より重くなります。保存のたびにその完了まで待つと、編集の応答がモデルや画像の大きさに左右されます。そこで保存時には、ZODBというPloneのデータベースに「このコンテンツを更新する」という仕事の記録だけを積みます。この置き場がoutboxです。

肝は、コンテンツ本体の変更とoutboxへの記録を同じトランザクションで確定することです。保存が取り消されれば仕事も取り消され、成功すれば仕事が残る。「保存できたのに検索への通知だけ失った」という隙間を小さくする設計です。outboxには本文やベクトルを詰め込まず、workerが処理するときにコンテンツを読み直します。

再試行できるように、更新の新しさを持たせる

workerはバックグラウンドの専用スレッドです。本文の更新 (upsert)、権限の更新 (security_update)、削除 (delete) を処理し、削除と権限変更を先に進めます。本文の特徴計算が混み合っている、見せてよい範囲の更新を優先したいからです。

ZODBとLanceDBの更新は、1つのトランザクションでは巻き戻せません。LanceDBには書けたのに仕事の完了記録が確定しなければ、同じ仕事をもう一度処理することがあります。各行にはZODBのコミット識別子を使ったgeneration (世代) を持たせ、新しい世代だけを書き込みます。古い仕事が遅れて届いても新しい状態を上書き

きしない、at-least-onceの配送を前提とした工夫です。

複数のプロセスがあるときは、期限付きの使用権であるleaseをZODBで取得したworkerが書き込みます。TTLは60秒で、処理のたびに延長します。ただし旧workerを強制排除するfencing tokenは無く、完全な二重書き込み防止ではありません。この制約は末尾の運用上の限界にもつながります。

実装を読む：全体のデータフローとoutbox

2. 意味が近くても、閲覧できなければ返さない

検索エンジンには権限の写しがありますが、更新は非同期です。公開していたページを非公開に戻した直後、検索側には古い権限が残るかもしれません。そのため、検索インデックスの判定だけで結果を返さず、2段階で確認します。

05 / PERMISSIONS & FRESHNESS

「似ている」と「見せてよい」を両方確かめる

画像検索では、権限に加えて「その画像の版か」も確認します。

1. 候補の前に絞る

LanceDB の ACL
公開期間・削除状態

2. 今の権限で確認

Plone の ZCatalog
現在の利用者で検索

3. 画像フィールドの権限

親ページだけでなく
画像そのものを見せてよいか

4. 画像の版を照合

差し替え前の特徴で
新しい画像を返さない

1・2は文章検索でも実施。3・4は画像検索で追加し、通過した候補だけを返します。

LanceDBでの事前絞り込み、現在のPlone権限による最終確認、画像フィールドの閲覧権限、画像の版の一致という確認の段階。

最初はLanceDBのprefilterです。利用者のロールやユーザーに対応するACL、公開期間、削除状態、検索条件で候補を絞ります。次に候補のUIDを使い、現在の利用者としてZCatalogのrestrictedなsearchResultsを引き直します。ここで返らない候補は検索結果から落とします。

非公開化が検索側にまだ届いていなくても、最終確認で止まります。逆に、公開したのに検索側の権限がまだ厳しい場合は、反映されるまで見つからないことがあります。表示の遅れと情報の漏れを同じ問題にせず、現在の閲覧権限を最後に守る設計です。候補が減ったときは追加取得を試しますが、回数には上限があります。

使えないときにも、応答の意味を保つ

文章検索はLanceDBやベクトルが使えないとき、理由を付けてZCatalogへ戻る経路を持ちます。APIのlance.fallbackを見ると、別の検索方式で応答したことが分かります。画像検索には同じ意味の代替手段がないため、利用できないときは理由付きの503を返します。「結果が0件」と「検索機能が動いていない」を区別できます。

実装を読む：最終認可の実装

3. 検索対象に合わせて、データを分ける

同じページでも、キーワード検索は文書単位、意味の検索は本文の一部分単位、画像検索は画像単位で扱う方が自然です。そのためテーブルを分けています。異なる単位の結果を最後にコンテンツへまとめることが、次の順位の統合にも効いてきます。

table	1 行	定義
documents__<site> (rebuild の後は ...__g<N>)	コンテンツ 1 つ	backend/schema.py
chunks__<site>	vector 検索の chunk 1 つ	backend/chunk_schema.py
images__<site>__g<N>	画像 1 枚	backend/image_schema.py
terms__<site>__g<N>	「もしかして」の語 1 つ	backend/terms.py
自動タグ付けの語と、文 章の判定用の embedding	語 1 つ / 文書の窓 1 つ	backend/autotag_terms.py・ backend/autotag_passages.py

どのテーブルを検索で使うかは、Ploneの設定領域にあるmanifestが決めます。再構築中のテーブルを検索側がうっかり開かないための境界です。テーブル名のsite部分はサイトIDから正規化され、サイトごとのデータを区別します。

4. 文章検索：文字・意味・順位の3つの仕事

キーワード：広く候補を拾い、文字列で確かめる

全文検索のインデックスは1〜3文字のn-gramを使います。日本語の「本」のような1文字でも候補を拾い、英字の大文字・小文字や全角・半角などを正規化して照合します。日本語を形態素解析で単語に分ける方式ではありません。

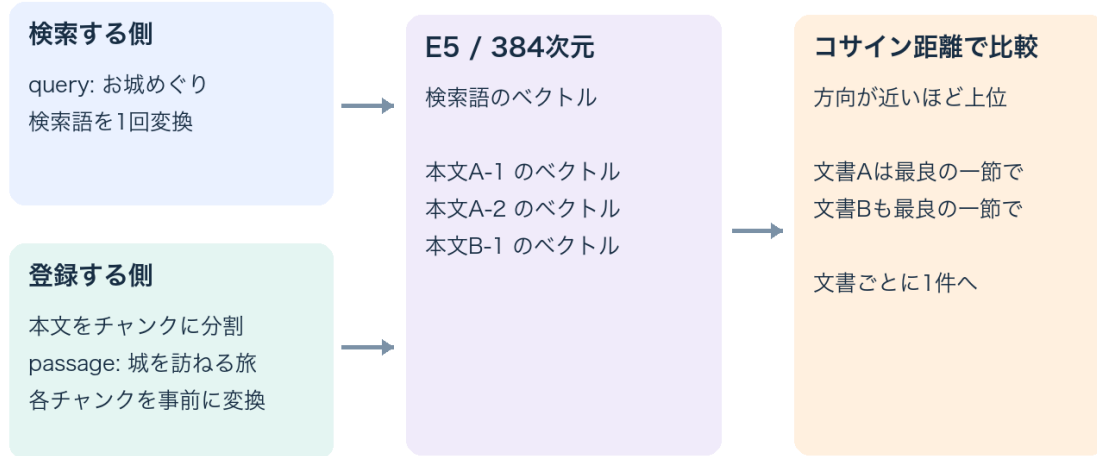
n-gramの一致だけでは候補が広すぎるため、肯定の語から候補を集めた後、正規化した本文に対してAND・OR・除外・フレーズの条件を確認します。題名・説明・本文の全文検索には、それぞれ3.0・1.5・1.0の重みを与えます。題名に書かれた語を重く扱いつつ、最後の条件照合で「本当に含むか」を確認しています。

ベクトル：文章を384個の数値へ写す

01 / TEXT EMBEDDINGS

文章を分け、意味の近さを比べる

検索語と本文は、同じ E5 モデルで384個の数値になります。



チャンクの数合計点にしないため、長い記事が単純に加点されるのを避けられます。

E5で検索語と本文チャンクを384次元のベクトルにし、コサイン距離で比較します。各文書では最も近いチャンクを代表にします。

ベクトル化 (embedding) は、文章を数値の並びへ変換する処理です。この実装ではmultilingual-e5-smallを使い、検索語も本文も384次元のベクトルになります。「城を訪ねる旅」と「お城めぐり」のように、字面が違ってても内容が近い文章を、近い方向へ写せることを利用します。ただし、意味の近さはモデルの判断であり、正しい答えを保証するものではありません。

モデルへの入力には役割があります。検索語には「query:」、本文には「passage:」を付けます。どちらも同じ文字列に揃えるのではなく、モデルが学習した検索語と本文の役割に合わせます。接頭辞を取り違えても数値は出てしまうため、エラーの有無だけでは品質の低下に気づきにくい部分です。実装にはこの入力契約を守るテストがあります。

長いページは、いちばん近い一節で代表させる

本文はモデルのtoken単位で区切ります。tokenは文字数とは異なるモデルの入力単位で、この実装は512 tokenから接頭辞分を引いた507 tokenの窓を使い、重なりは0です。各部分をchunk (チャンク) として事前にベクトル化します。検索時は検索語だけを1回変換し、チャンクとのコサイン距離を計算します。

コサインは数値ベクトルの向きの近さを見る尺度です。距離が小さいチャンクほど近いとみなし、文書ごとに最も近いチャンクを代表にします。合計点にすると長い文書ほどチャンクが多いだけで有利になるため、足し合わせません。どの一節が見つかったかは、検索結果の抜粋にも利用します。

このブランチのchunksには、近似近傍探索のANNインデックスを張っていません。権限などで絞り込んだ行を総当たりで比較します。「ベクトルDBを使っているから、必ず近似探索」というわけではなく、規模が大きくなると対象チャンク数が検索コストに直接効きます。

実装を読む：文書ごとに最良のチャンクを選ぶ

ハイブリッド：BM25とコサインを、順位でつなぐ

キーワード検索のBM25スコアと、ベクトル検索の類似度は、値の意味も尺度も異なります。そのまま足すと片方の数値の大きさに引っ張られます。ここで使うのがRRF (Reciprocal Rank Fusion)。各検索の「何位だったか」から点を作り、同じコンテンツの分を足します。

02 / RANK FUSION

点数の単位が違うなら、順位で合わせる

RRF：各順位 r の $1 / (60 + r)$ を足す。以下は説明用の仮の順位です。

キーワード順位

1位 記事A
2位 記事B
記事C は候補外

ベクトル順位

1位 記事B
2位 記事C
記事A は候補外

統合後の順位

1位 記事B 0.03252
2位 記事A 0.01639
3位 記事C 0.01613

$$\text{記事B} = 1 / 62 + 1 / 61 \div 0.03252$$

両方の上位にいる記事が上がる。片方だけの記事も候補に残る。

この値は順位を合成するための点数です。正しさの確率や一致率ではありません。

説明用の順位例。記事Bはキーワード2位とベクトル1位で、 $1/62+1/61$ により約0.03252。両方の上位にある記事が統合後に上がります。

計算は、各順位 r に対して $1 / (60 + r)$ 。片方に無い記事は、その側の加点がありません。図では記事Bがキーワード2位・ベクトル1位なので約0.03252、キーワード1位だけの記事Aは約0.01639です。両方で支持される記事が上がりやすく、片方でしか見つからない記事も残ります。この点数は正解確率ではありません。

キーワードはdocuments、ベクトルはchunksという別テーブルから来るため、先にUIDで文書単位へ揃えてからアドオン側でRRFを計算します。さらにページをめくるたび順位が動いて重複するのを避けるため、101件ずつの段に分けて統合し、ページはその1本の並びから切り出します。検索方式を混ぜる処理には、画面のページ分割まで含めた整合性が必要です。

実装を読む：RRFの式とページに依存しない順位

条件を厳密に守りたいとき

文章検索のAND・OR・フレーズ条件は、キーワード側の一致に適用されます。ベクトル側は意味の近さから候補を出すため、同じ字面を含まない結果もあります。厳密な文字条件で絞るならキーワードを選びます。「-語」による除外は文章の3方式すべてで適用され、意味で見つかった候補も、その語を本文に含めば除かれます。

5. Voltoから、検索処理へ届くまで

VoltoはReactの画面からplone.restapiを呼びます。このブランチでは、標準の@searchにuse_site_search_settingsを付けた要求を、サイトの検索設定に応じてLanceDBへ流せます。通常のコンテンツ選択や一覧ブロックまで一律に切り替える仕組みではありません。検索画面の関連度順と、標準カタログを使う用途を分けています。

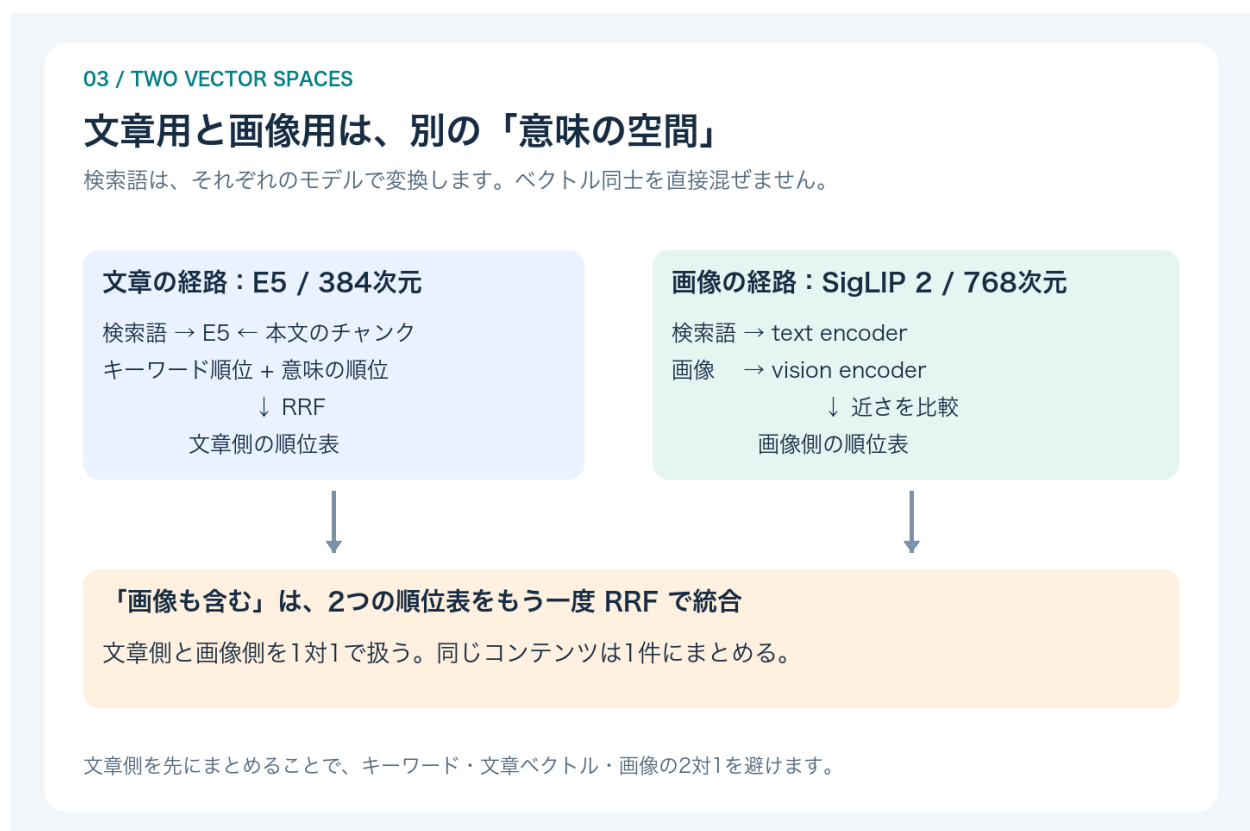
専用の@lance-searchではsearch_scopeをcontent・images・allから指定します。JSON APIの既定モードはkeywordなので、APIで比較するときはmodeを明示します。一方、対応する検索画面はベクトルが有効ならhybridを選ぶ経路があります。「同じ言葉を渡したのに結果が違う」ときは、入口とモードも確認する必要があります。

画面が自動でhybridを選び、ベクトルを利用できなかった場合はkeywordで引き直す経路があります。URLで利用者がモードを明示した場合は扱いが異なり、理由付きで従来の検索へ戻ります。日付順など、関連度以外の並べ替えも標準検索の役割です。

実装を読む：Volto連携の入口と適用条件

6. 画像検索：言葉と写真を同じ空間へ

画像には、文章用のE5とは別にSigLIP 2を使います。言葉を変換するtext encoderと、画素を変換するvision encoderが、同じ768次元の空間へ写します。「電車」という検索語のベクトルと、電車の写真のベクトルを直接比べられるので、タイトルがIMG_1234でも写真の内容を検索の手がかりにできます。



E5の384次元空間は文章用、SigLIP 2の768次元空間は言葉と画像用。別々の順位表を作り、画像も含む検索では順位を再統合します。

画像はEXIFの向きを反映し、透過を白へ合成してRGBに変換し、224×224へ縮めてモデルに渡します。この実装は縦横比を保持せず縮小します。前処理とモデルの組が変わると数値の意味も変わるため、モデル・前処理・

tokenizerなどからprofile_idを作り、異なるprofileの索引と検索モデルを混ぜないようにします。

写真の内容を毎回文章で説明してから検索する方式ではありません。索引を作る際に画像の特徴を計算して保存し、検索時には入力された言葉の特徴を計算します。画像を人が完全に理解するのと同じではないため、細かい文字、個数、専門的な物体名などをどこまで区別できるかは、実際の検索で確かめる必要があります。

重い画像推論を、CMS本体から分ける

画像のデコードとvision encoderの推論は、workerが起動する子プロセスで行います。子はZODB・ LanceDB・ Zopeを読み込まず、標準入出力のフレームで画像と結果を受け渡します。親が処理時間、メモリ、異常終了を監視するため、壊れた画像や大きなモデルの問題をCMSの仕事から分離しやすくなります。

画像の検索語をベクトル化するtext encoderは、検索を受けるプロセスで動きます。つまり「画像を登録するとき」と「言葉で検索するとき」では、動く場所もモデルの役割も違います。モデルはONNX Runtimeで実行し、画像モデルの取得はHTTP要求中に行わず、事前のCLIで準備・ 検証します。

差し替え前の写真が、検索結果に化けないようにする

画像検索ではPloneの最終認可に加えて、画像フィールド自体を見られるか、索引のsource_revisionが現在の画像の版と合うかを確認します。たとえば電車の写真を花の写真に置き換えた直後、古い「電車」の特徴で新しい花の画像を返してしまうのを防ぎます。

標準の抽出器が対象にするのは、Dexterityの画像フィールドやlead imageです。本文HTMLに現れるすべての外部画像を、そのまま収集する仕組みではありません。本文に表示したPloneのImageコンテンツは、そのImage自身のフィールドとして索引の対象になります。JPEG・ PNG・ 静止画WebPが対象で、HEICは追加対応が必要です。

文章と画像を混ぜるとき、重みが偏らない工夫

「画像も含む」(all) は、まず指定された方法で文章側の順位を作ります。その順位と、画像側で親コンテンツごとに最良の1枚を代表にした順位を、もう一度RRFで混ぜます。キーワード・ 文章ベクトル・ 画像を3列のまま混ぜると文章側が2票になりますが、2段階にすれば文章と画像を1対1で扱えます。画像を大量に持つコンテンツにも枚数だけで加点しません。

実装を読む：画像の前処理・ 推論・ 認可・ 統合

7. 検索の特徴を、タグ付けにも使う

自動タグ付けでは、管理者が登録した語もベクトルに変換し、画像や文章との近さを比べます。何でも自由な語を生成する方式ではなく、使いたい語彙の中から候補を選ぶ方式です。付与と候補のしきい値を分け、強く一致する語をタグに、迷う語を人が見直す候補にできます。

さらに「タグのお勧め」は、似たコンテンツに付いているタグから候補を作ります。モデルが計算する近さと、編集者が蓄えてきた分類をつなぐ使い方です。お勧めの計算でも、編集者が閲覧できるコンテンツだけを根拠にします。

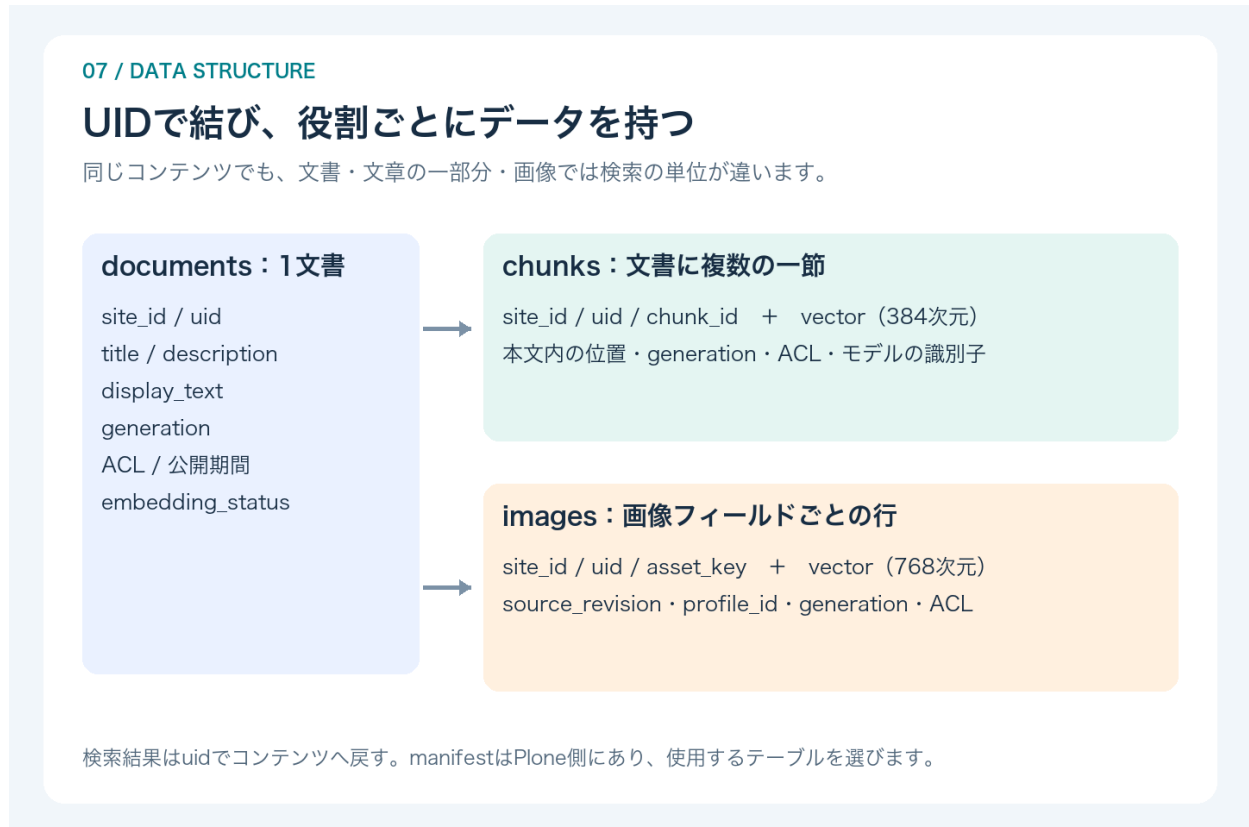
自動で付けた語、候補、編集者が外した語を記録するため、人が不要と判断した語を後で何度も付け直すのを避けられます。文章のタグ付けにはE5またはMiniLMを選べますが、文章検索用のモデルとは設定の役割が異なります。機能の有効・ 無効はサイトの管理設定に従います。

実装を読む：候補語・ しきい値・ 編集者の判断

アドバンス：内部の構造をコードと結び付ける

ここからは、検索結果の裏にどんな行と識別子があるのかを追います。重要なのは「何を同一とみなすか」と「どの時点の状態を使っているか」です。

A. 文書・チャンク・画像をつなぐキー



documentsは文書単位、chunksはsite_idとuidとchunk_id、imagesはsite_idとuidとasset_keyで区別。ACLと世代を各検索用データに持たせます。

documentsはコンテンツ1つの行です。タイトル、説明、表示用本文、正規化した検索用本文に加え、UID・公開期間・ACL・世代を持ちます。本文のハッシュが変わらなければフィルター列だけを更新でき、権限の変更で文章の特徴を毎回作り直す必要がありません。

chunksのキーはsite_id・uid・chunk_id。1つの文書に複数のチャンクが結び付き、vectorだけでなく本文中のsource_start・source_endを持ちます。これにより、ヒットを文書へまとめても「どの一節が近かったか」を失わず、抜粋へ戻せます。ACLを複製するのは、近さを計算する前に権限で対象を絞れるようにするためです。

imagesのキーはsite_id・uid・asset_key。asset_keyはmain:imageなど、どの画像フィールドかを表します。uidはその画像フィールドを持つコンテンツのUIDで、画像を本文に貼り付けた記事のUIDと常に同じではありません。「画像を持つコンテンツ」と「画像を参照する記事」は区別します。

実装を読む：チャンクのスキーマ

実装を読む：画像のスキーマとキー

B. 「世代」「画像の版」「モデルの版」は別の問いに答える

識別するもの	主な値	防ぎたい取り違え
更新の新旧	generation (ZODBのコミット識別子)	遅れて届いた古い更新で新しい状態を上書きする
検索先のテーブル	manifestが指すactiveのテーブル名	再構築途中のテーブルを検索してしまう
画像そのものの版	source_revision	差し替え前の特徴から新しい画像を返す
特徴計算の契約	profile_id (画像)、embedding_modelなど (文章)	異なるモデル・前処理の数値を同じ意味として比較する

すべてを「バージョン」と呼ぶと、何を検査しているかが曖昧になります。元データの新しさ、検索対象の切り替え、画像の実体、数値を作った条件は、それぞれ違う軸です。画像を差し替えたときにrevisionを、モデルを変更したときにprofileを確認することで、見た目では気付きにくい取り違えを防ぎます。

C. 「祭り」を文章と画像の両方で探すときの追跡

入口は@lance-searchのsearch_scope=all。文章側はdocumentsのキーワード候補とchunksの意味の候補を集め、文書単位にまとめて最終認可し、指定モードの順位を作ります。画像側はSigLIP 2で作った検索語の特徴を使ってimagesを探し、親コンテンツ・フィールド・画像の版を確認します。

次に画像側を親コンテンツごとの順位へまとめ、文章側の順位とRRFで統合します。serializerは画面へ渡す結果を組み立て、matched_byにtext・image・bothのどれかを示します。ここを見ると、同じ検索結果でも「文章から来たのか」「画像から来たのか」を説明できます。これは内部のベクトル値を利用者へそのまま見せることとは別です。

実装を読む：本文と画像の統合処理

D. Volto対応の工夫：標準の抽出器が動く環境を作る

このブランチで扱った問題の1つは、「Voltoで書いた本文が標準検索では見つかるのに、workerが作った索引には題名しか入らない」というものでした。本文抽出を独自に書き直すのではなく、標準の抽出器を呼ぶ環境に原因がありました。

plone.restapiのブロック本文抽出は、objとrequestからIBlockSearchableTextというアダプターを探します。バックグラウンドのworkerには通常のHTTPリクエストがなく、global requestが空だと必要なアダプターを見つけられません。そこでindexing/context.pyは作業中のrequest・site・security managerを整え、終了時には例外があっても元へ戻します。

Plone標準の拡張ポイントを再利用できるからこそ、その標準処理が前提にするコンテキストも揃える必要がある。これは、ライブラリを呼ぶだけでは済まないCMS統合の具体例です。workerは索引を作るためにsystem userで読みますが、検索結果は現在の閲覧者として認可します。この2つの権限の役割も分離されています。

実装を読む：workerのコンテキスト設定と後片付け

E. @searchの応答を、自前の別APIに変えない

browser/rest_search.pyのRestSearchHandlerは、plone.restapiのSearchHandlerを継承します。ハンドラー内のcatalog呼び出しを包み、条件に合う検索だけLanceDBに流します。最終認可済みのcatalog brain (カタログが持つメタデータの結果オブジェクト) を標準serializerへ渡すため、Voltoには従来のsummary形式が返ります。portal_catalogそのものを全用途で交換する実装ではありません。

切り替えにはuse_site_search_settingsというキーの存在を使います。値が1か0かでon/offを決めるものではありません。object browserも検索語を送るため、SearchableTextがあるだけで切り替えると編集用の選択画面まで巻き込むからです。範囲外の並べ替えや絞り込みなどは、標準のカatalogへ委ねます。

この部分の面白さは、Voltoを全面改造して独自の応答形式へ合わせるのではなく、標準APIの変換・シリアライズの流れを利用したまま検索処理へ接続している点です。機能を増やす境界を狭く保つことが、既存の編集体験との両立につながります。

実装を読む：RESTの標準ハンドラーへの接続

8. コードを読み進めるための地図

検索の入口はquery/service.py、RRFはquery/hybrid.py、文書単位への集約はquery/aggregate.py、権限の最後の確認はquery/authorizer.pyです。まずこの4つを読むと、モデルの推論とCMSの責務がどこでつながるかを追えます。以下のパスはsrc/c2/search/lancedb/からの相対パスです。

package	役割
backend/	LanceDB の読み書き (embedded.py)、table の定義、prefilter の述語、fake の backend
indexing/	outbox・worker・lease・writer・抽出・chunk・vectorize・画像の同期と推論
query/	service・compiler・最終認可・候補の取り直し・hybrid・vector の mode・画面の mode・snippet
embedding/	text の embedding の provider (E5・MiniLM)
image/	画像の asset・形式・推論の子 process と protocol・検索語の埋め込み
maintenance/	manifest と設定・status・保守の操作・自動タグ付け
browser/	画面と JSON の view (検索・画像検索・設定ページ・似た画像・タグのお勧め・TinyMCE)
security/	権限の token

9. 動くことと、品質を保つこと

検索は結果が返っただけでは正しさ確かめられません。接頭辞の取り違い、権限チェックの抜け、ページ送りの重複は、それぞれ別の検証が必要です。tests/test_quality_gates.pyでは、検索がmanifestの選んだテーブルだけを開くことなど、実装の構造そのものも検査します。

開発時の入口はmake test、make lint-check、make typecheck、make ciです。CIで確認されている組は資料上Plone 6.2.1とPython 3.13。実際の画像モデルを使うONNX検証は別の記録として管理し、モデルや前処理を変えたのに検証記録が追いついていない状態も検査対象にしています。

10. どこまでできて、何を測るべきか

意味の近さは正解の保証ではなく、言い換えを拾うと同時に、関連はあるが欲しかったものではない結果も増えます。キーワードとベクトルを同じ検索語で比べ、「欲しい記事が見つかるか」「上位に何が混ざるか」を両方確認することが大切です。

この実装はチャンクにも画像にもベクトル用のANNインデックスを張りません。件数が増えたときは文書数だけでなく、本文の長さ、チャンク数、画像数、権限による絞り込み後の件数を見ます。最初のモデル読み込みと、モデルを保持した後の検索時間も分けて測る必要があります。

キーワードは部分一致で、誤字を自動で同一視するfuzzy検索ではありません。ICUを併用すると語のまとまりによる順位の補正ができますが、文字列の一致条件そのものを置き換えるものではありません。総件数や候補の補充にも上限があり、画面上の件数は常に全体の正確な件数を表すわけではありません。

書き込み側のleaseには、期限切れの旧writerを拒むfencing tokenがありません。複数台の同時書き込みを完全に防ぐ保証として扱わず、writerの配置と状態を運用で確認する必要があります。モデルの精度だけでなく、更新・権限・障害・規模まで含めて設計するところに、CMSへ検索を組み込む面白さがあります。

管理者向け：状態の読み方と再構築の考え方

利用者向け：検索を比較しながら体験する

持ち帰って試す

検索設計の読解演習

ひとつの検索結果を、元のPloneコンテンツ、検索用の文書、本文のチャンク、画像の特徴へ対応付けてみましょう。順位を作る責任と、閲覧を許可する責任がどの層にあるかを図にすると、検索基盤とCMSを統合する意味が見えてきます。

記録すること

入力した言葉 / 対象コンテンツ / 検索方式 / 結果の違い / 次に確かめたいこと

この演習の見出しはPDF本文に含まれています。サイトの検索で見出しを入力すると、PDF本文が検索対象になっていることも確かめられます。