01

Contextual Retrieval のご紹介

— Introducing Contextual Retrieval —

公開日	2024-09-19
原題	Introducing Contextual Retrieval
著者	Anthropic Engineering Team
原文	https://www.anthropic.com/engineering/contextual-retrieval
翻訳	Claude（機械翻訳／Anthropic）
編集	2026-04-09

Contextual Retrieval のご紹介

AI モデルを特定の文脈で有用に使おうとすると、背景知識へのアクセスが必要になることがよくあります。たとえば、カスタマーサポート用のチャットボットは自分が配属されたビジネス固有の知識を必要としますし、法律分析ボットは過去の膨大な判例を知っている必要があります。

開発者は通常、**Retrieval-Augmented Generation（RAG）** を使って AI モデルの知識を強化します。RAG は、ナレッジベースから関連情報を検索してユーザーのプロンプトに追加し、モデルの応答を大幅に強化する手法です。問題は、従来の RAG が情報をエンコードする時点で文脈を取り除いてしまうことにあり、結果としてナレッジベースから関連情報をうまく検索できないケースがよく発生します。

本稿では、RAG の検索ステップを劇的に改善する手法を紹介します。この手法は「Contextual Retrieval（文脈化検索）」と呼び、**Contextual Embeddings（文脈化埋め込み）** と **Contextual BM25（文脈化 BM25）** という 2 つのサブテクニックを使います。これにより検索失敗率を 49% 削減でき、reranking（再順位付け）と組み合わせれば 67% 削減できます。これらは検索精度の大幅な改善であり、下流タスクの性能向上にも直結します。

私たちのクックブックを使えば、あなた自身の Contextual Retrieval を Claude で簡単にデプロイできます。

まずは単にプロンプトを長くすることについて

時には、最もシンプルな解が最良です。ナレッジベースが 20 万トークン（およそ 500 ページ分）より小さいなら、そもそも RAG などを使わずにナレッジベース全体をモデルへのプロンプトに丸ごと含めてしまえばよいのです。

数週間前、私たちは Claude 向けにプロンプトキャッシュをリリースしました。これにより、このアプローチがはるかに高速かつコスト効率の良いものになります。開発者は、API 呼び出しの間で頻繁に使うプロンプトをキャッシュでき、レイテンシを 2 倍以上削減し、コストを最大 90% 削減できます（詳しくはプロンプトキャッシュのクックブックを参照してください）。

とはいえ、ナレッジベースが大きくなるにつれ、よりスケーラブルな解決策が必要になります。そこで登場するのが Contextual Retrieval です。

RAG 入門: より大規模なナレッジベースへのスケーリング

コンテキストウィンドウに収まらないような大きなナレッジベースに対しては、RAG が一般的な解になります。RAG はナレッジベースを次の手順で前処理します。

1. ナレッジベース（文書の「コーパス」）を小さなテキストの塊（チャンク）に分割する。通常は数百トークンを超えないように。
2. これらのチャンクを埋め込みモデルで、意味をエンコードしたベクトル埋め込みに変換する。
3. 埋め込みを、意味的類似度で検索できるベクトルデータベースに格納する。

実行時、ユーザーがクエリを入力すると、ベクトルデータベースはクエリとの意味的類似度に基づき最も関連するチャンクを探します。そして最も関連するチャンクが、生成モデルへのプロンプトに追加されます。

埋め込みモデルは意味的な関係を捉えるのが得意ですが、重要な**厳密一致**を見逃すことがあります。幸い、そのような状況を補える古典的な手法があります。**BM25**（**Best Matching 25**）は、語彙的なマッチングで厳密な単語やフレーズの一致を見つけるランキング関数です。一意な識別子や技術用語を含むクエリに特に有効です。

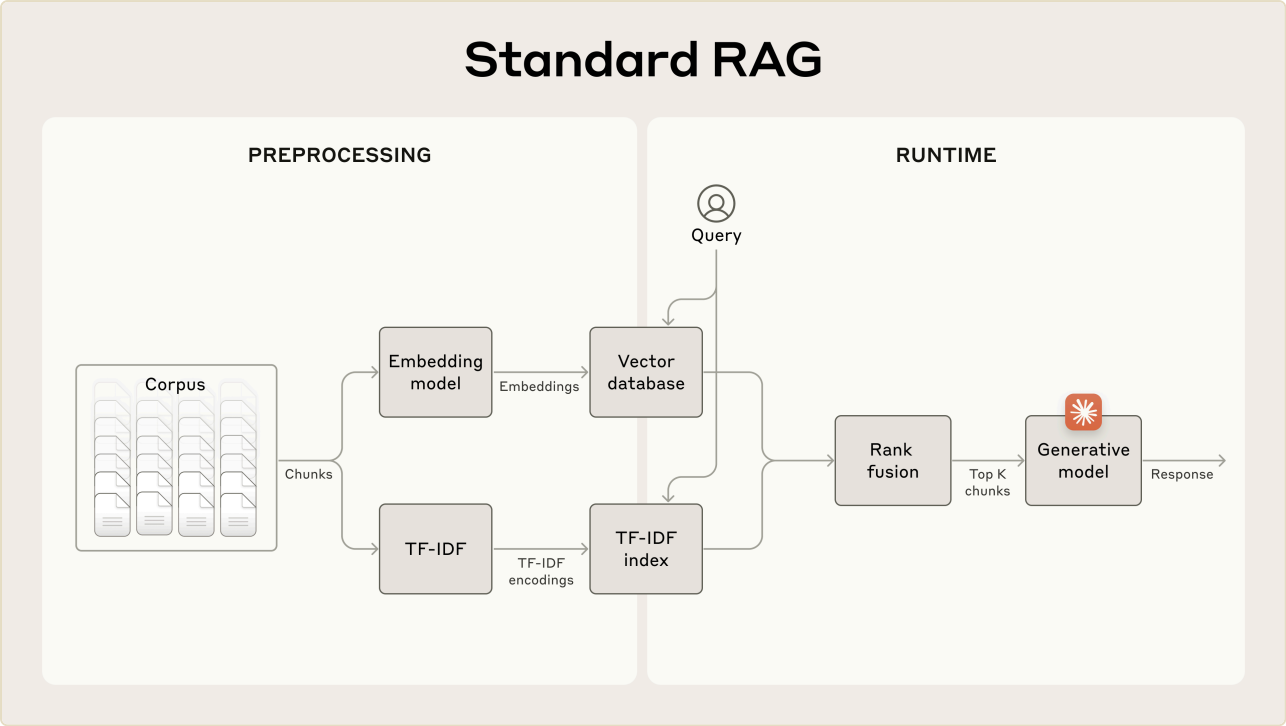
BM25 は **TF-IDF**（**Term Frequency-Inverse Document Frequency**）の考え方を土台にしています。TF-IDF は、ある単語がコレクション中の文書にとってどれほど重要かを測ります。BM25 はこれを、文書の長さを考慮し単語頻度に飽和関数を適用することで洗練し、よくある単語が結果を支配してしまうのを防ぎます。

セマンティック埋め込みが失敗する場面で BM25 が成功する例を挙げましょう。ユーザーが技術サポートのデータベースに対し「エラーコード TS-999」と問い合わせたとします。埋め込みモデルはエラーコード全般の情報を返せても、「TS-999」そのものの厳密一致を見逃すかもしれません。BM25 はこの具体的な文字列を探し、該当するドキュメントを特定します。

RAG は、次のように埋め込みと BM25 を組み合わせることで、より正確に該当チャンクを取り出すことができます。

1. ナレッジベース（文書の「コーパス」）を小さなテキストの塊に分割する。通常は数百トークンを超えないように。
2. これらのチャンクに対して TF-IDF エンコーディングと意味的埋め込みを作成する。
3. BM25 で厳密一致に基づくトップチャンクを探す。
4. 埋め込みで意味的類似度に基づくトップチャンクを探す。
5. (3) と (4) の結果を rank fusion 手法で結合・重複除去する。
6. 上位 K 件のチャンクをプロンプトに追加して応答を生成する。

BM25 と埋め込みモデルの両方を活用することで、従来型の RAG システムは厳密な語句マッチと広い意味的理解のバランスを取り、より包括的で正確な結果を提供できます。



埋め込みと BM25 の両方を使って情報を検索する標準的な RAG システム。TF-IDF（term frequency-inverse document frequency）は単語の重要度を測り、BM25 の基礎となる。

このアプローチを使えば、単一プロンプトに収まりきらないような巨大なナレッジベースにも、コスト効率よくスケールできます。しかし、こうした従来型の RAG には大きな制約があります。文脈をしばしば破壊してしまうのです。

従来型 RAG における文脈の謎

従来型 RAG では、効率的な検索のために文書を小さなチャンクに分割するのが一般的です。このアプローチは多くのアプリケーションでうまく動きますが、個々のチャンクが十分な文脈を持たない場合には問題を引き起こします。

たとえば、米国 SEC 提出書類のような金融情報のコレクションをナレッジベースに埋め込んでおり、こんな質問を受け取ったと想像してください。*「ACME Corp の 2023 年第 2 四半期の売上成長率はいくつでしたか?」*

関連するチャンクには次のような文が含まれているかもしれません。*「当社の売上高は前四半期比で 3% 成長した。」* しかしこのチャンク単体では、どの会社のことなのかも、どの期間のことなのかも指定されていません。これでは正しい情報を検索するのも活用するのも困難です。

Contextual Retrieval の紹介

Contextual Retrieval は、埋め込みを作る前（「Contextual Embeddings」）と BM25 インデックスを作る前（「Contextual BM25」）に、各チャンクの先頭にそのチャンクを説明する文脈を付与することで、この問題を解決します。

先ほどの SEC 提出書類の例に戻りましょう。チャンクは次のように変換されます。

```
original_chunk = "The company's revenue grew by 3% over the previous quarter."

contextualized_chunk = "This chunk is from an SEC filing on ACME corp's performance
in Q2 2023; the previous quarter's revenue was $314 million. The company's revenue g
rew by 3% over the previous quarter."
```

なお、検索改善のために文脈を利用する手法は過去にも提案されてきました。たとえば、汎用的な文書サマリーをチャンクに追加する方法（私たちも実験しましたが、得られる効果はごく限定的でした）、仮想文書埋め込み（Hypothetical Document Embedding）、サマリーベースのインデクシング（評価しましたが性能は低かったです）などです。これらは本稿で提案する手法とは異なります。

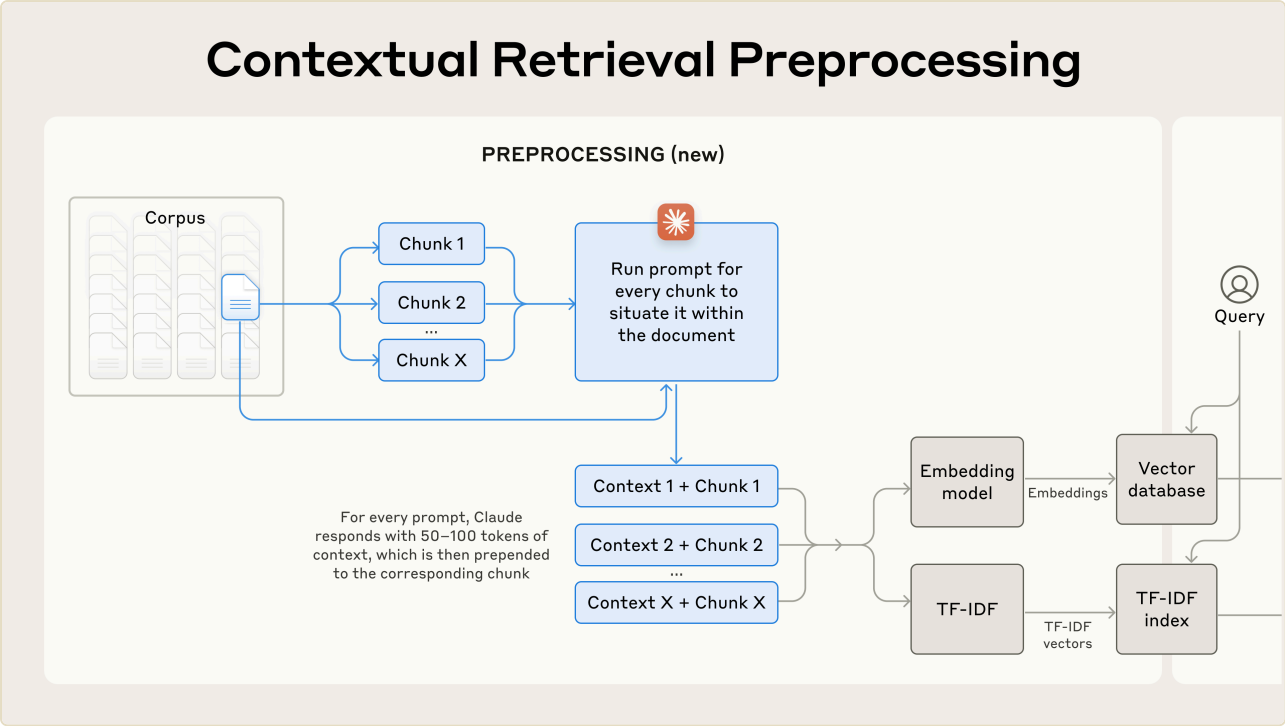
Contextual Retrieval の実装

もちろん、数千・数百万のチャンクに人手で注釈を付けるのは現実的ではありません。Contextual Retrieval を実装するには、Claude を使います。私たちは、全文書の文脈を踏まえて各チャンクを説明する簡潔な文脈をモデルに生成させるプロンプトを書きました。チャンクごとの文脈生成には、次のような Claude 3 Haiku 向けのプロンプトを使用しました。

```
<document>
{{WHOLE_DOCUMENT}}
</document>
Here is the chunk we want to situate within the whole document
<chunk>
{{CHUNK_CONTENT}}
</chunk>
Please give a short succinct context to situate this chunk within the overall docume
nt for the purposes of improving search retrieval of the chunk. Answer only with the
succinct context and nothing else.
```

生成される文脈テキストは通常 50〜100 トークン程度で、これをチャンクの先頭に付与した上で埋め込み生成および BM25 インデックス作成を行います。

実際の前処理フローは次のようになります。



Contextual Retrieval は検索精度を改善する前処理手法である。

Contextual Retrieval を試してみたい方は、私たちのクックブックから始めてみてください。

プロンプトキャッシュで Contextual Retrieval のコストを下げる

先ほど紹介した特別なプロンプトキャッシュ機能のおかげで、Contextual Retrieval は Claude なら低コストで実現できます。プロンプトキャッシュがあれば、チャンクごとに参照文書を渡す必要はありません。一度だけキャッシュに文書をロードし、その後はキャッシュ済みコンテンツを参照するだけです。800 トークンのチャンク、8k トークンの文書、50 トークンの文脈生成用指示、チャンクあたり 100 トークンの文脈という前提で、**文脈化されたチャンクを生成する 1 回限りのコストは、文書トークン 100 万あたり 1.02 ドル**になります。

方法

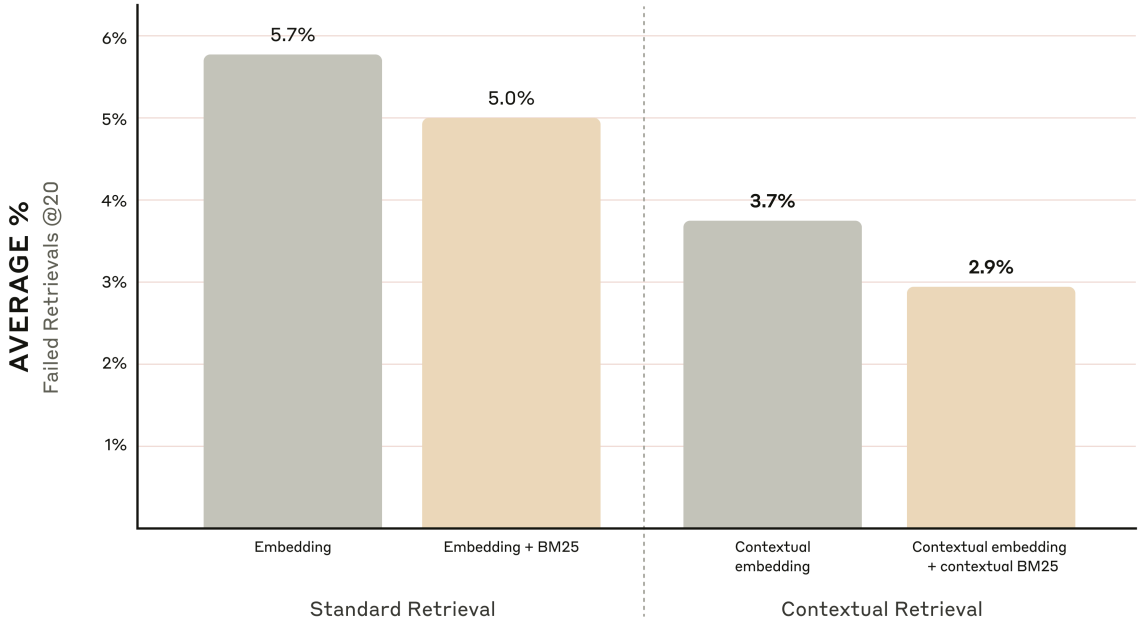
私たちはさまざまなナレッジドメイン（コードベース、小説、ArXiv 論文、科学論文）、埋め込みモデル、検索戦略、評価指標にわたって実験を行いました。各ドメインで使用した質問と回答の例は付録 IIに一部収録しています。

下図のグラフは、最も性能の良かった埋め込み構成（Gemini Text 004）とトップ 20 チャンク検索で、全ナレッジドメインにわたる平均性能を示したものです。評価指標には「1 − recall@20」を用いています。これは、関連文書のうちトップ 20 チャンクに入らなかった（検索失敗した）割合を測ります。完全な結果は付録にあります。私たちが評価したすべての埋め込み+ソースの組み合わせで、文脈化は性能を改善しました。

性能改善

実験で分かったのは次のことです。

- **Contextual Embeddings はトップ 20 チャンクの検索失敗率を 35% 削減しました**(5.7% → 3.7%)。
- **Contextual Embeddings と Contextual BM25 を組み合わせると、トップ 20 チャンクの検索失敗率は 49% 削減しました**(5.7% → 2.9%)。



Contextual Embedding と Contextual BM25 を組み合わせると、トップ 20 チャンクの検索失敗率は 49% 削減される。

実装上の考慮点

Contextual Retrieval を実装する際に覚えておきたい考慮点がいくつかあります。

1. **チャンク境界**: 文書をチャンクに分ける方法を考えましょう。チャンクサイズ、チャンク境界、チャンクのオーバーラップは検索性能に影響します。
2. **埋め込みモデル**: Contextual Retrieval は私たちがテストしたすべての埋め込みモデルで性能を改善しましたが、モデルによって恩恵の大きさは異なります。Gemini と Voyage の埋め込みが特に効果的でした。
3. **ドメイン特化の文脈生成プロンプト**: 私たちが提示した汎用プロンプトでもうまく機能しますが、自分のドメインやユースケースに合わせたプロンプト(たとえばナレッジベース内の他文書にしか定義されていない重要用語の用語集を含めるなど)を用意すれば、さらに良い結果を得られるかもしれません。

4. **チャンクの個数**: コンテキストウィンドウに多くのチャンクを入れるほど、関連情報を含める可能性は上がります。ただし情報が多すぎるとモデルは気が散ってしまうため、限度があります。私たちは 5、10、20 チャンクを試し、この中では 20 チャンクを渡すのが最も性能が良いと分かりました（比較は付録参照）。とはいえ自分のユースケースで実験する価値はあります。

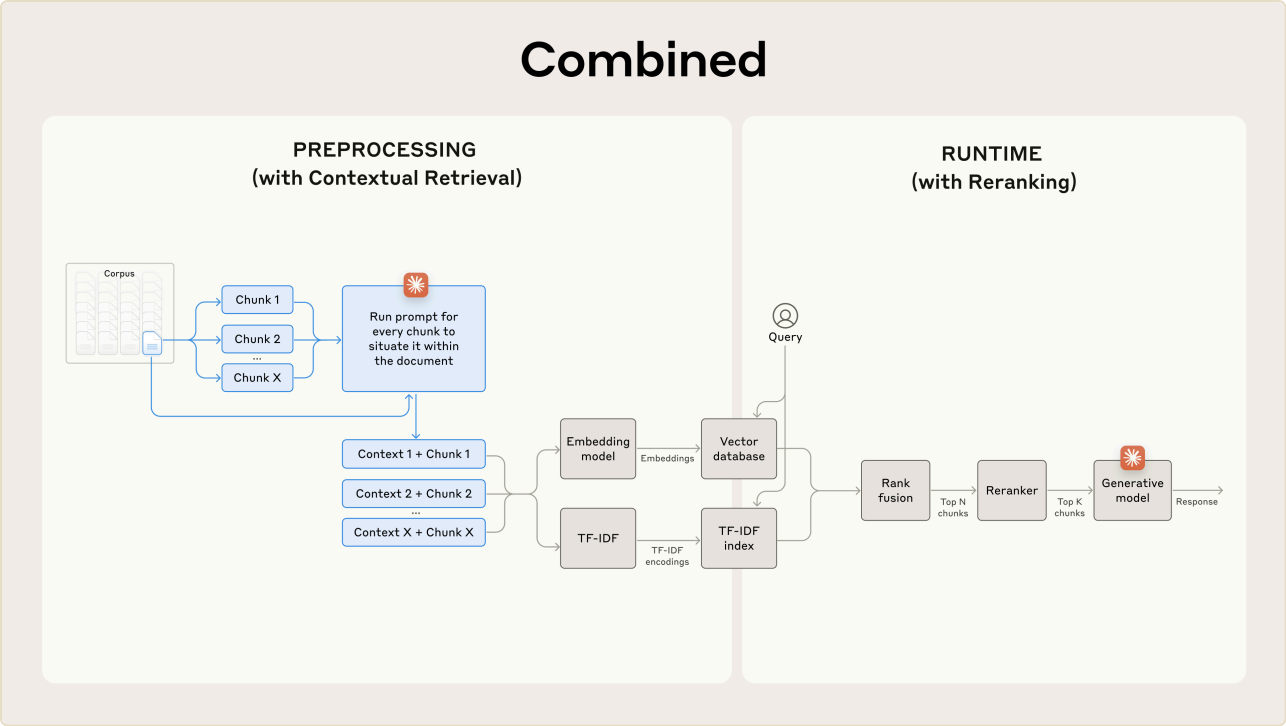
常に eval を回すこと: 応答生成は、文脈化されたチャンクを渡しつつ、どれが文脈でどれがチャンクかを区別してモデルに提示することで改善できる可能性があります。

Reranking でさらに性能を高める

最後のステップとして、Contextual Retrieval を別のテクニックと組み合わせ、性能をさらに引き上げることができます。従来型 RAG では、AI システムがナレッジベースを探索して、潜在的に関連するチャンクを取得します。大きなナレッジベースでは、この初期検索で数十から数百のチャンクが返ってきて、関連度や重要度もまちまちです。

Reranking（再順位付け） は、最も関連度の高いチャンクだけをモデルに渡すために使われる一般的なフィルタリング手法です。Reranking によってより良い応答が得られ、モデルが処理する情報量が減ることでコストとレイテンシも下がります。主要な手順は次のとおりです。

1. 初期検索で潜在的に関連するトップのチャンクを取得する（私たちはトップ 150 件を使用）。
2. トップ N 件のチャンクを、ユーザーのクエリとあわせて reranking モデルに通す。
3. Reranking モデルで、各チャンクにプロンプトに対する関連度と重要度のスコアを付け、上位 K 件を選ぶ（私たちはトップ 20 件）。
4. 上位 K 件のチャンクを文脈としてモデルに渡し、最終結果を生成する。

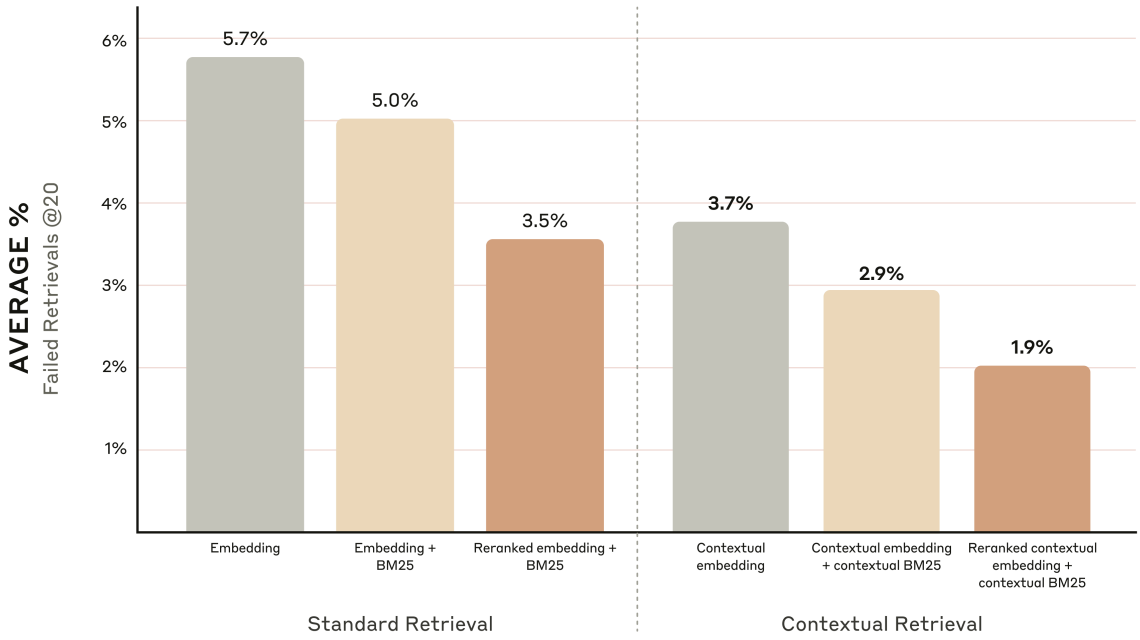


Contextual Retrieval と Reranking を組み合わせて検索精度を最大化する。

性能改善

Reranking モデルは市場にいくつかあります。私たちは Cohere の reranker でテストしました。Voyage も reranker を提供していますが、時間の都合でテストできていません。実験では、さまざまなドメインにおいて reranking ステップを加えることで検索がさらに最適化されました。

具体的には、**Reranked Contextual Embedding + Contextual BM25 はトップ 20 チャンクの検索失敗率を 67% 削減しました**（5.7% → 1.9%）。



Reranked Contextual Embedding と Contextual BM25 の組み合わせは、トップ 20 チャンクの検索失敗率を 67% 削減する。

コストとレイテンシの考慮

Reranking で重要な考慮点は、レイテンシとコストへの影響です。特に多数のチャンクに対して reranking を行う場合はそうです。Reranking は実行時に追加ステップを入れるため、reranker がすべてのチャンクを並列にスコアリングするとしても、どうしても少量のレイテンシが増えます。より良い性能を得るために多くのチャンクを reranking するか、より低レイテンシ・低コストを得るために少なく reranking するかの、本質的なトレードオフがあります。自分のユースケースでさまざまな設定を試し、適切なバランスを見つけることをお勧めします。

結論

私たちは多数のテストを走らせ、上述した手法（埋め込みモデル、BM25 の使用、Contextual Retrieval の使用、reranker の使用、取得するトップ K の総数）のあらゆる組み合わせを、さまざまな種類のデータセットで比較しました。分かったことを要約します。

1. Embeddings + BM25 は、埋め込み単体より良い。
2. 私たちがテストしたなかで最良の埋め込みは Voyage と Gemini だった。
3. 上位 20 チャンクをモデルに渡す方が、上位 10 や 5 より効果的。
4. チャンクに文脈を加えることで、検索精度は大幅に改善する。

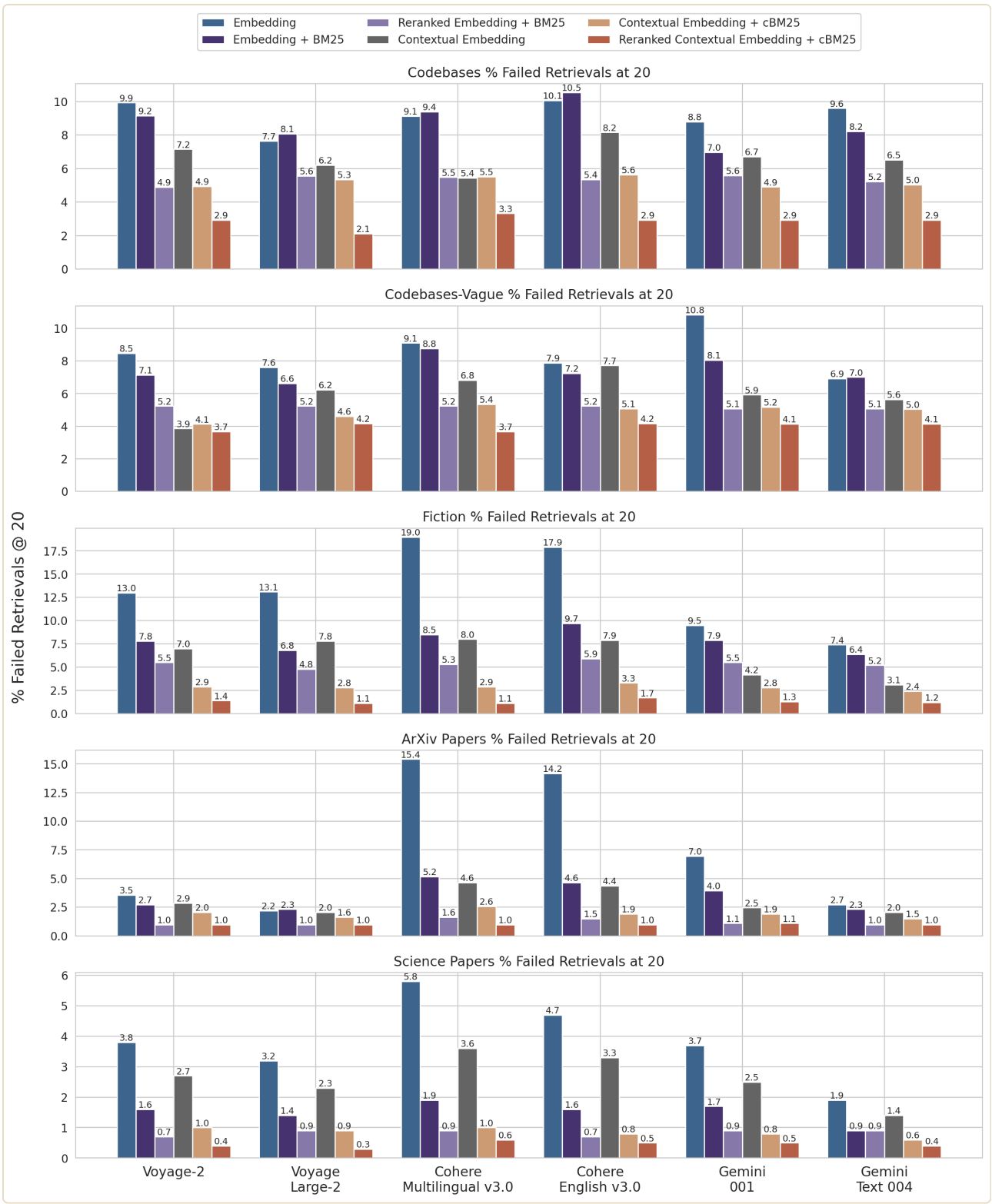
5. Reranking あり は Reranking なしより良い。
6. **これらの恩恵はすべて積み重ねが効く。**性能改善を最大化するには、Contextual Embeddings（Voyage または Gemini）＋ Contextual BM25 ＋ Reranking ステップ ＋ 20 チャンクをプロンプトに追加、という組み合わせが良い。

ナレッジベースを扱うすべての開発者の方に、私たちのクックブックを使ってこれらのアプローチを試し、新たな性能水準を解き放つことを勧めます。

付録 I

以下は、データセット別、埋め込みプロバイダ別、埋め込みに加え BM25 を併用するか否か、Contextual Retrieval を使うか否か、reranking を使うか否かに関する Retrievals @ 20 の結果の内訳です。

Retrievals @ 10 および @ 5 の内訳と、各データセットの質問・回答例については付録 IIを参照してください。



データセットおよび埋め込みプロバイダ横断の 1 − recall @ 20 の結果。

謝辞

研究と執筆: Daniel Ford。重要なフィードバックをくれた Orowa Sikder、Gautam Mittal、Kenneth Lien、クックブックを実装してくれた Samuel Flamini、プロジェクトをコーディネートしてくれた Lauren Polansky、本稿を形作ってくれた Alex Albert、Susan Payne、Stuart Ritchie、Brad Abrams に感謝します。