CDD2 の具体的応用としての RAG

制約解釈にもとづく知識選択アーキテクチャの一考察

Y. Matsuda

2026-03-13
2026-03-22

概要

Retrieval-Augmented Generation（RAG）は，大規模言語モデルに検索機構を接続することにより，外部知識を利用した応答生成を可能にする代表的技術である。しかし，実運用の観点から見ると，RAG は単一の方式ではなく，制約をどのように導入するかによって，その内部構造も利用者にもたらす価値も大きく変化する。本稿は，CDD2 の具体的応用例として RAG を取り上げ，制約をほとんど用いない RAG，一般的な制約付き RAG，そして CDD2 型 RAG を比較する。

議論の中心は二つある。第一は，それぞれの RAG がどのようなアーキテクチャを持ち，実装上どこに重心を置くかという点である。第二は，その違いが最終的に利用者にどのような価値の差として現れるかという点である。制約をほとんど使わない RAG は，検索と生成を単純に接続することで高い利便性を実現するが，信頼性や一貫性に弱い。一般的な制約付き RAG は，アクセス制御やガードレールを加えることにより安全性と運用品質を高めるが，なお回答の意味成立条件そのものを扱うわけではない。これに対して CDD2 型 RAG では，制約は単なる補助条件ではなく，どの知識を，どの順序で，どの意味で採用するかを決める中核となる。

$$\text{Query} \to \text{Retrieval} \to \text{Generation} \to \text{Answer}$$

という処理連鎖ではなく，

$$\text{Query} \to \text{Constraint Interpretation} \to \text{Knowledge Selection} \to \text{Structured Answer}$$

という意味選択の過程へと再定義されるという点にある。この再定義によって，RAG は単なる検索補助付き生成系ではなく，制約解釈を伴う知識選択系として理解されるべきであり，利用者には単なる利便性を超えた「説明可能な信頼」を提供しうる。

なお、付録として追加した Cognitive RAG は、CDD2 ではカバーしない「(状況に応じた）判断を行うための選択肢」を与える。

1　例：企業ナレッジ検索 AI としての RAG

議論を具体化するため，本稿では企業内で用いられるナレッジ検索 AI を例に取る。想定するのは，規程類，契約書，社内マニュアル，FAQ，運用メモなどを横断的に検索し，自然言語で回答す

る AI である。利用者はたとえば次のような質問を行う。

Q: この契約の解約条件は何ですか。

A: 契約書第 12 条によれば，解約には 30 日前の書面通知が必要です。

一見すると，これは単純な質問応答に見える。しかし実際には，この短い Q&A の背後に多くの問題が潜んでいる。たとえば，契約書には改訂版と旧版があるかもしれない。法務部が保持する正式版と，営業部が運用している参考版が矛盾しているかもしれない。ある契約類型では第 12 条より別紙の例外規定が優先されるかもしれない。また，質問者の所属によって閲覧可能な文書範囲が異なることもある。つまり，利用者が求めているのは単に「それらしい答え」ではなく，「どの知識に基づき，どの優先順位と規則に従ってそう答えたのか」が担保された回答である。

この例は，RAG の比較にきわめて適している。なぜなら，同じ質問に対しても，制約をどこにどう入れるかによって，内部アーキテクチャも最終的な回答の意味も大きく変わるからである。以下では，この企業ナレッジ検索 AI の例を通じて，三種類の RAG を順に検討する。

1.1 制約をほとんど使わない RAG

知識状況

- 契約書.pdf：「解約には 30 日前の書面通知が必要」
- FAQ.pdf：「実務では柔軟に対応することもある」
- 運用メモ.pdf：「緊急時は即時解約を認めた事例あり」

典型的な出力例

A 契約書によれば、解約には 30 日前の書面通知が必要です。ただし、実務上は柔軟に対応される場合や、緊急時には即時解約が認められるケースもあります。

出来ない部分

- どの文書が支配的ルールか決定できない。そのため「原則」と「例外」を区別できない。
- その結果、文書間の矛盾（たとえば契約書では解約には 30 日を要する、一方メモでは即時 OK とある）を解決できない。
- 答えを与えているわけではない。判断できない。

制約をほとんど使わない RAG は，最も素直な構成を持つ。すなわち，

$$\text{User} \to \text{Retriever} \to \text{LLM} \to \text{Answer}$$

という流れである。実装上は，まずクエリを埋め込み表現へ変換し，ベクトルデータベースから上位 k 件の文書片を取得し，その取得結果を LLM に入力し，自然言語としての回答を得るという構成になる。したがって，ここでの主要な関心は，検索精度，埋め込み精度，プロンプトの書き方，そして LLM の要約能力に向かう。

企業ナレッジ検索 AI の例で言えば，利用者が「この契約の解約条件は何ですか」と質問すると，

システムは契約書の断片，FAQ の断片，場合によっては関連の薄い運用文書まで含めて上位候補として取り出し，それらをまとめて LLM へ渡す。そして LLM は，その断片群をもとにもっともらしい解答を生成する。ここでは「正式版の契約書を優先するべきである」「FAQ は参考情報にすぎない」「同名文書の新版が旧版に優先する」といった規則は，あっても暗黙的な期待にとどまり，構造の中核にはなっていない。

この場合，制約は存在するとしてもほぼ暗黙的である。たとえば「なるべく正確であるべきだ」「可能なら根拠を用いるべきだ」といった期待は存在するが，それが明示的な構成要素として実装に強く組み込まれているわけではない。そのため，この方式の利点は明快である。構成は軽く，立ち上げは速く，PoC を作りやすく，小規模なナレッジ検索であれば実用上十分な場合も多い。

しかし，この単純さはそのまま弱点でもある。検索された文書片は，ほぼそのまま意味資源として LLM へ渡されるため，無関係文書の混入，権限外文書の混入，旧版と新版の混在，根拠のない生成，さらには矛盾する文書群をそのまま混ぜたもっともらしい回答の生成といった問題が起きやすい。たとえば，正式な契約書の第 12 条では「30 日前通知」と規定している一方，営業向け FAQ では「実務上は 14 日前で足りる場合がある」と書かれていたとする。制約をほとんど使わない RAG は，この二つを同列の検索結果として取り込み，「通常は 14 日前ですが，契約によって 30 日前の場合もあります」といった曖昧な答えを返しうる。ここでは，どちらが正式な根拠かという区別が，アーキテクチャ上ほとんど扱われていない。

利用者の観点から見れば，この方式の価値は主として利便性にある。速く，使いやすく，導入しやすく，とにかく答えが返ってくる。しかしその反面，信頼性，一貫性，説明可能性，組織知の統制という観点では弱い。したがって，利用者にとっては「便利ではあるが，重要業務にそのまま使うには不安が残る」という位置づけになる。

擬似コード

```
answer(query):
    docs = retrieve(query)
    response = generate_with_llm(query, docs)
    return response
```

1.2 一般的な制約付き RAG

典型的な出力例

A 契約書第 12 条によれば、解約には 30 日前の書面通知が必要です。

改善された部分

- 関係ない文書が混ざらない。
- 他部署の情報が含まれない。
- 古い文書が出にくい。

出来ない部分

- 依然、情報の優先順位が決定されない。したがって次のような回答が出る。

  A 契約書では 30 日前通知が必要です。一方で FAQ では柔軟対応が可能とされています。また、メモでは緊急時の即時解約も示されています。
- つまり「原則」と「例外」を区別できない。
- 矛盾（「30 日必要」と「即日 OK」）を解決できない。
- 何を条件とするか、いわゆる「意味の選択問題」は未解決のまま

企業利用や組織利用を前提とすると，RAG に何らかの制約を組み込む必要が生じる。ここでいう制約とは，アクセス制御，根拠文書の必須化，新版優先，フィルタ条件，回答フォーマット制約，ガードレールなどである。この場合，アーキテクチャは通常，

$$\begin{aligned}&\text{User} \rightarrow \\ &\textbf{Query Filter} \rightarrow \text{Retriever} \rightarrow \\ &\textbf{Document Filter} \rightarrow \text{LLM} \rightarrow \\ &\textbf{Output Validator} \rightarrow \text{Answer}\end{aligned}$$

のように拡張される。

すなわち，基本 RAG の前段・中段・後段に制約層が付加される。前段では，ユーザ権限に応じて検索範囲を制限し，部門別インデックスを切り替え，禁止語や危険質問を判定する。中段では，取得文書の重複除去，新版優先，メタデータにもとづくフィルタリング，信頼文書のみの通過といった処理が行われる。後段では，引用の有無を確認し，禁止出力を除去し，所定の JSON や定型形式に整形する。

企業ナレッジ検索 AI の例では，利用者の質問に対して，まずその利用者が法務文書にアクセス可能かどうかが前段でチェックされる。次に検索された候補から旧版を除外し，最新版の契約書だけを残す。さらに回答後には，「契約書第 12 条」といった根拠が含まれているか，禁止された断定表現がないか，所定の出力形式になっているかが確認される。これにより，「誤って閲覧権限のない文書を参照する」「古い版に基づいて答える」「根拠をまったく示さずに断定する」といったリスクはかなり減少する。

ここで重要なのは，この段階においても中心はなお

$$\text{Retrieval} + \text{Generation}$$

にあるという点である。制約は処理を制御するルールとして働いているが，それは検索と生成の本体の周囲に置かれた品質保護層である。したがって実装の重心は，フィルタ設計，ガードレール設計，メタデータ管理，ポリシー適用，出力検査といった部分に置かれる。これは企業導入においては非常に現実的であり，実際，権限逸脱の低減，誤引用の抑制，回答形式の安定化，古い文書の混入防止，組織導入の容易化といった大きな効用をもたらす。

利用者にとっても，この方式は単なる利便性に加えて安全性と運用品質を提供する。つまり「便利な AI」から「安全に答える AI」へと一段進む。しかし，なお限界は残る。なぜなら，ここでの

制約はまだ「答えを囲うもの」であって，「答えの意味を作るもの」ではないからである。企業ナレッジ検索 AI の例で言えば，正式契約書の第 12 条では「30 日前通知」と規定されている一方，別紙の例外条項では「特定類型については即時解約が可能」とされているかもしれない。また，法務部通達では「原則として契約書本文優先」とある一方，実務マニュアルでは「緊急案件では別紙優先」と書かれているかもしれない。一般的な制約付き RAG は，これらを個別のフィルタや優先ルールで部分的に扱うことはできても，「この質問に対してどの規範が優先されるべきか」「どの例外が適用されるべきか」という意味選択の中心問題を十分には扱えない。

擬似コード

```
answer(query, user):
    filtered_query = apply_query_constraints(query, user)
    docs = retrieve(filtered_query)
    docs = filter_documents(docs, user)
    response = generate_with_llm(query, docs)
    response = validate_output(response, docs, user)
    return response
```

1.3 CDD2 型 RAG

回答例A

契約書第 12 条に基づき、解約には 30 日前の書面通知が必要です。緊急時には例外的に即時解約が認められる場合がありますが、原則として契約条項が優先されます。

制約付き RAG から変わった点

- 質問の意味空間が定まる。つまり「これは契約解釈問題であり、法的優先順位がある。そして根拠は条文ベース、例外は補助情報」が制約される。
- 検索が「意味」検索になる。つまり通常の RAG が似てる文書を取ってくるのに対し、CDD2 では制約に合う文書（contract.pdf は必須、FAQ は補助、memo は条件付き）を探索する。
- 証拠が構造的になる。つまり、次の 4 つ構造 Claim, Exception, Priority, Condition が明示的に扱われる。（Claim: 解約には 30 日前通知が必要（契約書）、Exception: 緊急時は即時解約の可能性（メモ）、Priority: 契約書 > 運用メモ、Condition: 例外は条件付き）

CDD2 型 RAG において本質的なのは，制約を単なるアクセス制御や出力制御としてではなく，どの知識を，どの順序で，どの意味で採用するかを決める中核として扱う点である。すなわち，制約は検索条件，フィルタ条件，出力条件を超えて，意味選択条件となる。

このとき，アーキテクチャは

User →

Constraint Interpreter → Constraint-aware Retrieval →

Evidence Structuring → LLM →

Constraint-based Validation → Answer

のように変化する。

Constraint Interpreter は，質問をそのまま検索語として扱うのではなく，その質問がどの知識領域に属するか，どの規範が優先されるべきか，何を根拠と見なすか，矛盾時にはどの規則で裁定するかを先に決定する。企業ナレッジ検索 AI の例で言えば，「この契約の解約条件は何ですか」という問いに対して，まずそれが契約解釈の問いであり，契約書本文，別紙，法務通達，FAQ，運用メモのうちどれを上位根拠とみなすかを決める。たとえば，「正式契約書本文を第一根拠とし，別紙は契約類型が一致する場合のみ優先し，FAQ は補足情報にとどめる」といった解釈方針がここで確定される。言い換えれば，この層は質問の意味空間を制約にもとづいて定める。

次に Evidence Structuring は，取得した文書片を単に並べるのではなく，主張，根拠，例外，優先順位，依存関係，矛盾関係として再構造化する。たとえば，契約書本文第 12 条は「原則 30 日前通知」という主張，別紙は「特定類型では即時解約可能」という例外，法務通達は「契約書本文優先」という優先規則，FAQ は「営業現場でのよくある誤解」を示す補助情報として整理される。この段階で，検索結果は単なる上位 k 件の文書断片ではなく，制約にもとづいて秩序づけられた証拠構造へと変換される。

最後に Constraint-based Validation は，単に危険な文でないかを確認するだけではなく，この回答がどの制約に基づいて成立しているか，優先規則に沿っているか，例外条件を落としていないか，知識間の矛盾を未解決のまま隠していないかを検証する。たとえば，最終回答が「原則として解約には 30 日前の書面通知が必要です。ただし，別紙に定める特定類型では即時解約が可能です」となったとき，その答えが本文優先・例外条件付与という制約に整合しているかが確認される。もし矛盾が解消されない場合には，曖昧さを隠した断定ではなく，「本文と別紙の適用関係の確認が必要です」と明示することが求められる。

この段階に至ると，実装の中心はもはや単なる RAG ではない。より正確には，**制約解釈器つき知識選択系**である。したがって，実装上は少なくとも明示的な制約モデルが必要になる。たとえば，公式文書は FAQ より優先され，FAQ は社内メモより優先される，新版は旧版より優先される，法務規定は部門ローカル運用より優先される，例外規定には適用条件がある，といった知識上の制約を明示的に持たなければならない。また，単なるベクトル検索だけでは足りず，内部では依存，上位下位，例外，競合，適用条件といった知識関係を扱う必要がある。さらに，回答生成の前処理として，どの根拠を中心に据えるか，どの例外を添えるか，どの矛盾を保留として示すかを決める工程が必要となる。

したがって，実装の重心は「検索の精度改善」から「意味成立条件の管理」へと移る。この移動こそが CDD2 型 RAG の特徴である。検索精度が重要でなくなるのではない。そうではなく，検索

精度に加えて，**何を正当な回答と見なすかを制約に基づいて決める層**が中心化するのである。企業ナレッジ検索 AI の文脈では，これは「契約に関する質問に対して，単に契約らしい断片を探して答える」のではなく，「契約解釈における規範体系に従って知識を選び，回答を組み立てる」ことを意味する。

擬似コード

```
answer(query, user, context):
    constraints = interpret_constraints(query, user, context)
    docs = retrieve_with_constraints(query, constraints)
    evidence = structure_evidence(docs, constraints)
    draft = generate_with_llm(query, evidence)
    answer = validate_by_constraints(draft, evidence, constraints)
    return answer
```

2 利用者価値の変化

利用者の観点から見たとき，三者の違いはさらに明確になる。制約をほとんど使わない RAG は，速く，使いやすく，導入しやすく，とにかく答えが返るという点で便利であるが，信頼の根が弱い。一般的な制約付き RAG は，そこに安全性と運用品質を付け加えるため，利用者にはより大きな安心感を与える。しかし，CDD2 型 RAG が提供する価値は，さらに一段深い。

企業ナレッジ検索 AI の例に戻れば，利用者が本当に必要としているのは，「30 日前です」とだけ答える AI ではない。正式文書を見ているのか，例外条項を考慮したのか，法務ルールと営業 FAQ のどちらを優先したのかが分かる AI である。CDD2 型 RAG では，利用者は単に「速い」「安全」「それらしい」応答を得るのではない。なぜその答えなのかが分かり，組織ルールや知識の優先順位に沿っており，矛盾や例外を隠さず，重要業務で使いやすく，運用の中で回答品質が安定するという価値を得る。利用者の印象も，「AI が勝手に答えている」から「組織知のルールに従って答えている」へと変わる。ここで生じるのは，単なる利便性や安全性ではなく，**説明可能な信頼**である。

A 参考：Cognitive RAG

当該者の「状況」が考慮される。たとえば

- コンプライアンス違反の疑いあり
- 法務確認は未実施
- 緊急性は中程度

回答例

現時点では、即時解約は推奨されません。
契約書第 12 条に基づき、解約には 30 日前の書面通知が必要です。例外として、重大なコンプライアンス違反が確認された場合には即時解約が認められる可能性がありますが、その場合でも法務部門の承認が前提となります。
現在は違反が「疑い」の段階であり、法務確認も未実施であるため、即時解約の条件は満たされていません。
まずは法務部門への確認を行い、違反の確定および緊急性の評価を行うことを推奨します。

CDD2 との大きな違い

- **当該者の状況に即し、質問者に判断選択肢を提供する**

擬似コード

```
answer(query, user, context):
    situation = collect_situation(query, user, context)
    constraints = interpret_constraints(query, situation)
    docs = retrieve_with_constraints(
        query,
        constraints,
        sources=["contract.pdf", "policy.pdf", "faq.pdf", "memo.pdf"]
    )
    evidence = structure_evidence(docs, constraints)
    state = reconstruct_state(situation, evidence, constraints)
    affordances = determine_affordances(state, constraints)
    plan = choose_response_or_action(state, affordances, constraints)
    answer = generate_with_llm(query, evidence, plan)
    answer = validate_by_constraints(
        answer,
        evidence,
        affordances,
        constraints
    )
    return answer
```