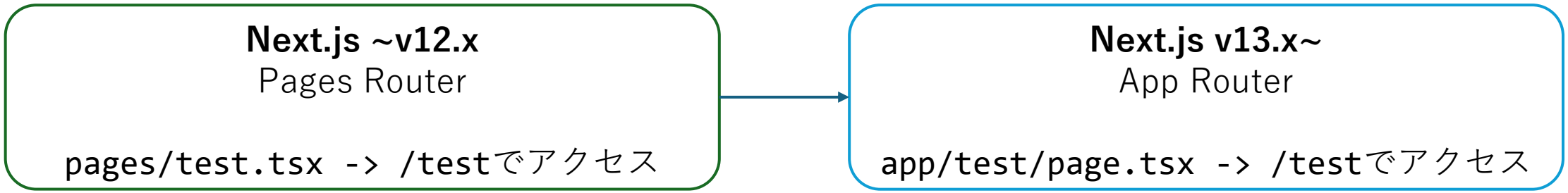
生成AIを活用した
ソフトウェア依存関係の自動更新

総合理工学研究科工学専攻 電子情報システム工学分野
23W2046G　小松 凌都

モジュールアップデートの課題

ソフトウェアプロジェクトにおけるモジュールアップデートには以下のような課題点を抱えている。

- 他のモジュールとのバージョン互換性の確保
- 破壊的変更への対応
- 脆弱性への対応



破壊的変更の例: Next.jsにおけるルーティング方法の変更

Dependabot

GitHubが提供している
依存関係を安全かつ最新状態に保つ自動化ツール

課題点（実装面への対応）

package.jsonなどの依存関係を管理するファイルのみ更新

破壊的変更には対応できない ⟶ ユーザによる手動修正

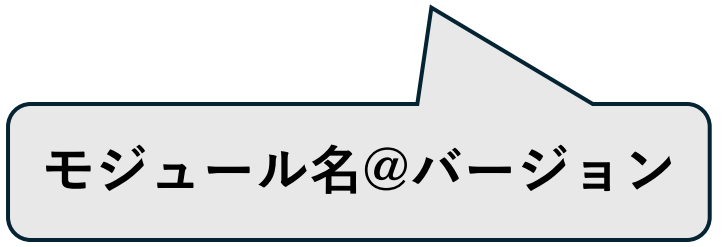
Denoについて

Node.jsの開発者Ryan Dahlが開発したJavaScriptランタイム

特徴

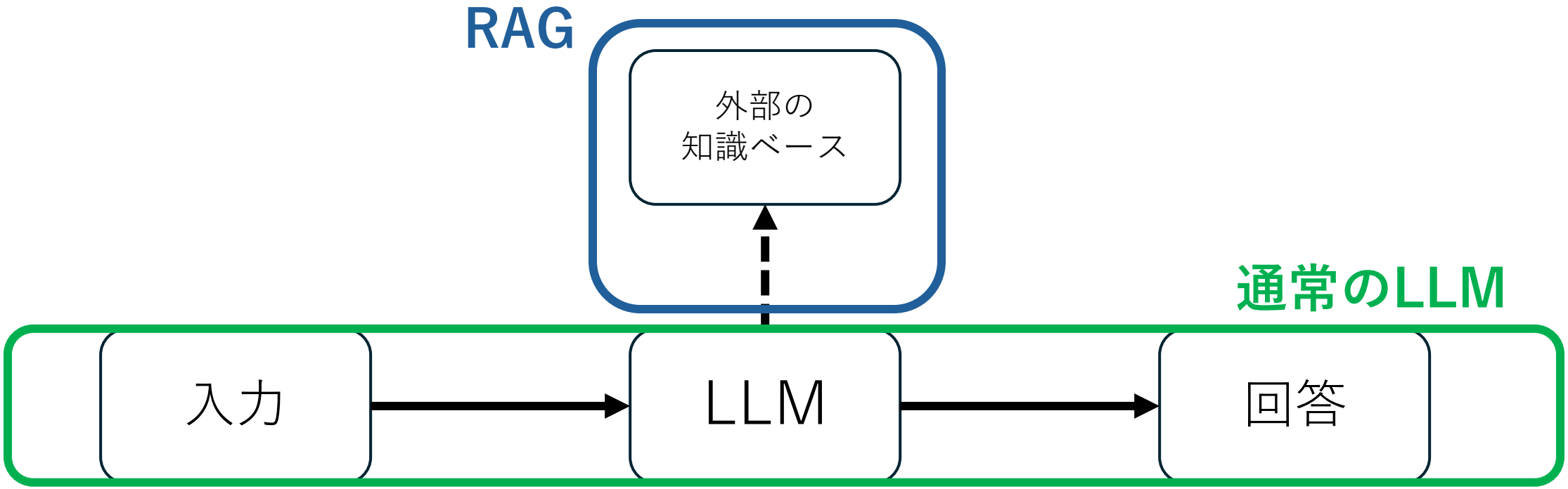
- デフォルトでTypeScriptが実行可能
- モジュールをURLで直接指定してインポート

```
import { Application } from “https://deno.land/x/oak@v12.6.1/mod.ts”
```



RAG（検索拡張生成）

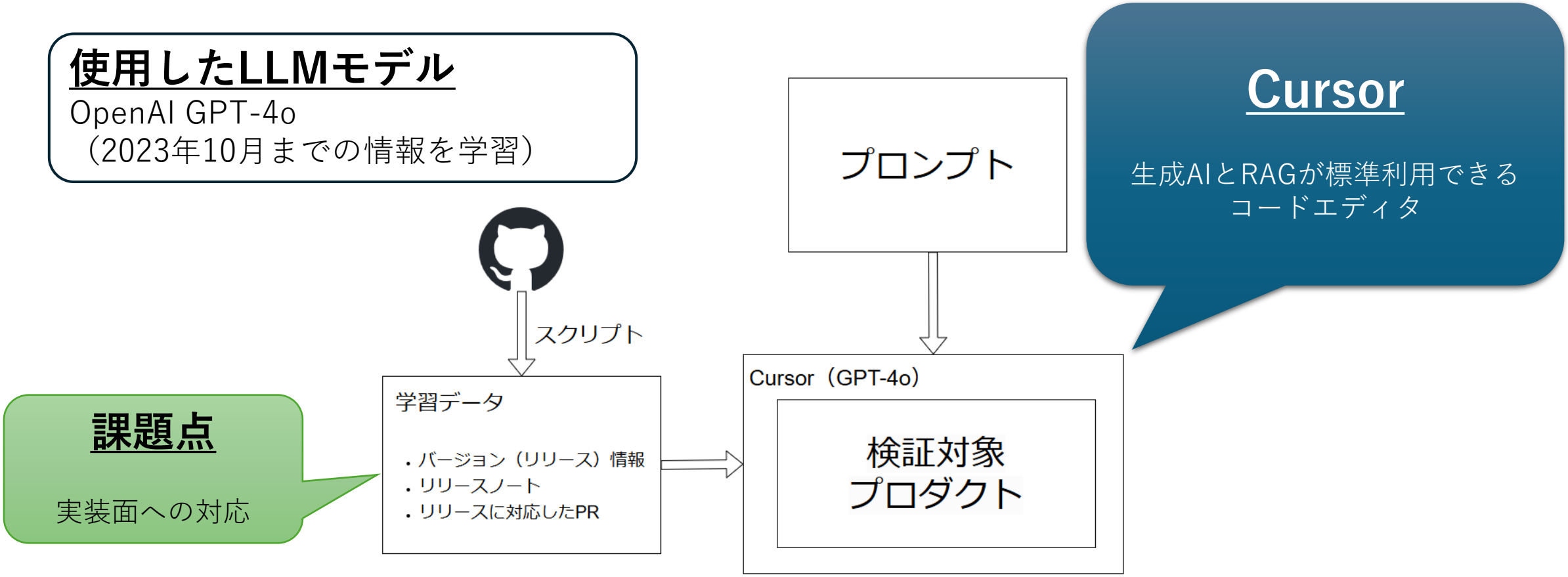
RAGでは外部の知識ベースを参照し、LLMの回答生成過程に組み込むことで、LLMが苦手とするドメイン特化の情報やモデル学習期間後の情報への回答がより正確になり、生成結果の根拠を確認することができる。



提案手法

Dependabotの課題点に対応するため、
以下の手法を提案する。

使用したLLMモデル
OpenAI GPT-4o
（2023年10月までの情報を学習）



Research Question

RQ1（提案手法にてプロダクトのアップデートはできるか）

1. アプリは実行可能であるか？
2. 各モジュールの最新版にアップデート出来ているか？

RQ2（与えたデータはアップデートでどのように利用されるか）

1. 出力結果ではどのデータを参照しているか？
2. 与えたデータを正確に解釈できているか？

【実験方法】
Cursor上に検証用プロダクトを用意

使用モジュール・ランタイム

fresh
- Deno向けのWebフレームワーク

preact
- freshで推奨されているUIモジュール

preact-render-to-string
- preactコンポーネントを HTMLにレンダリングする

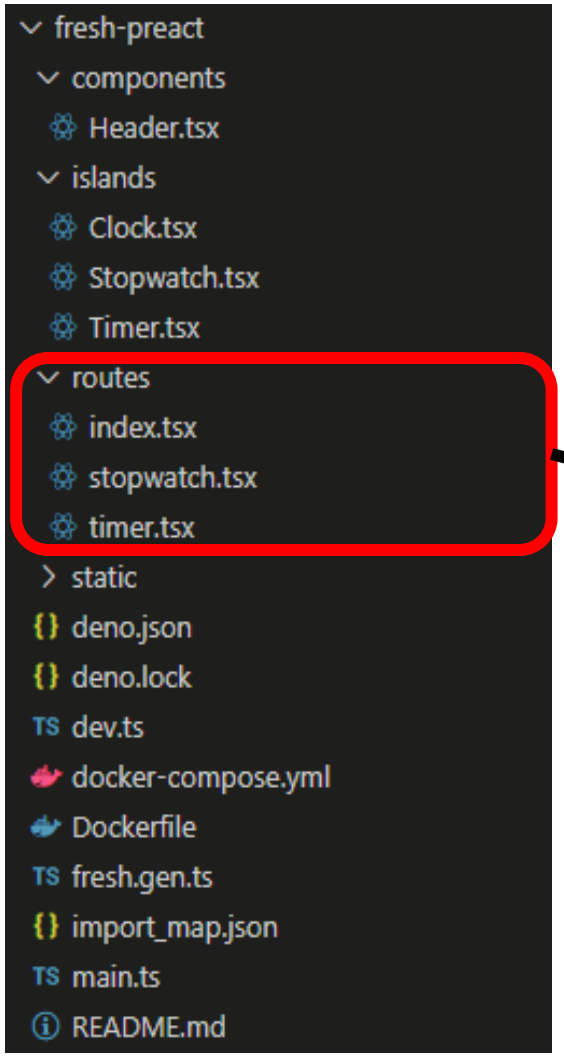
Deno
- TypeScriptをデフォルトサポートする JavaScriptランタイム

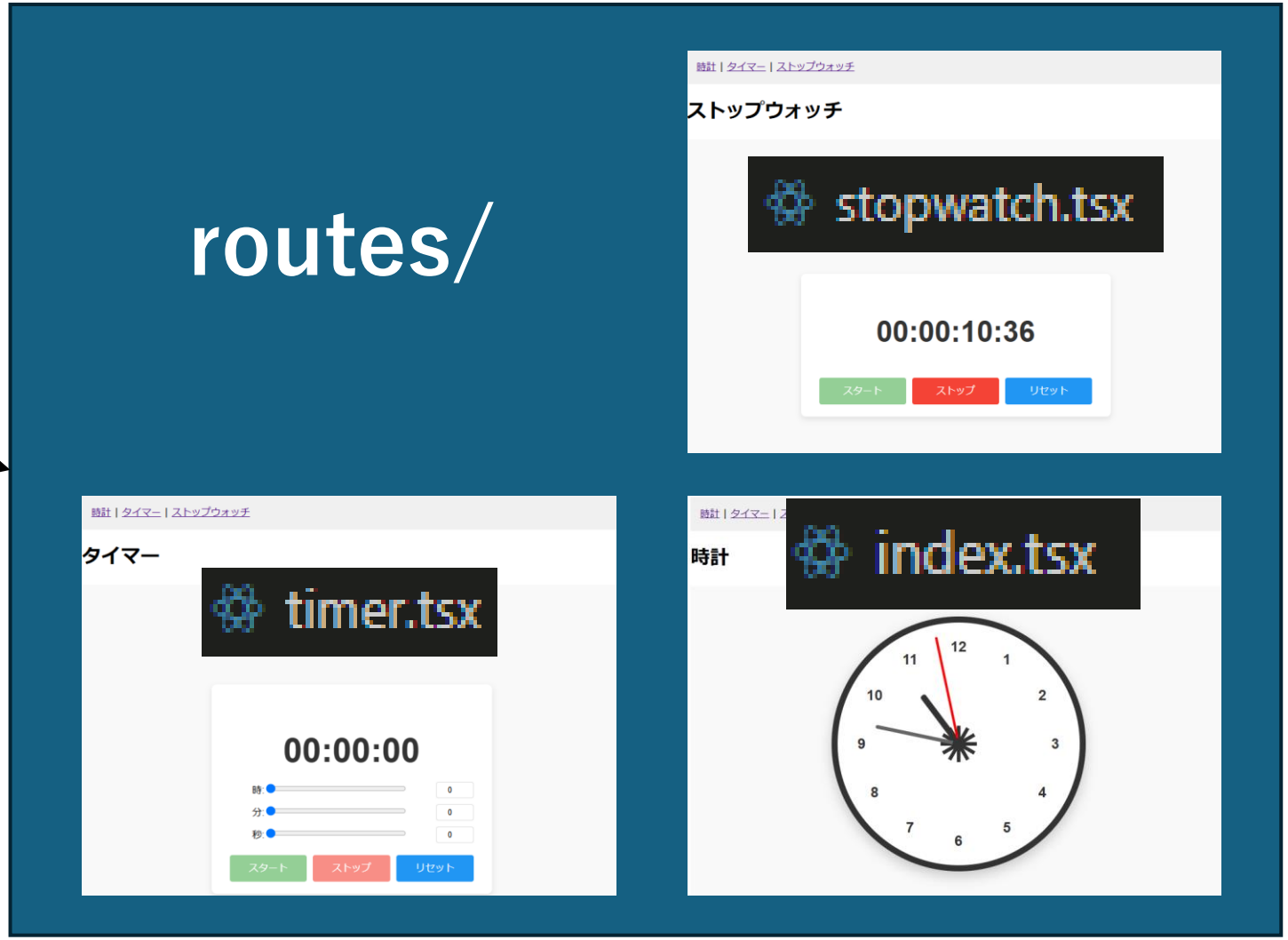
バージョン

更新前: モデル学習中期間の最新バージョン
更新後: 学習後の最新版（2024/11/30時点）

	更新前（リリース日）	更新後（リリース日）
fresh	1.4.3 (2023/09/06)	1.7.3 (2024/10/14)
Deno	v1.37.1 (2023/09/07)	v2.1.2 (2024/11/28)
preact	10.17.1 (2023/08/09)	10.25.0 (2024/11/22)
preact-render-to-string	6.2.1 (2023/08/10)	v6.5.11 (2024/09/15)

作成した検証用プロダクトについて

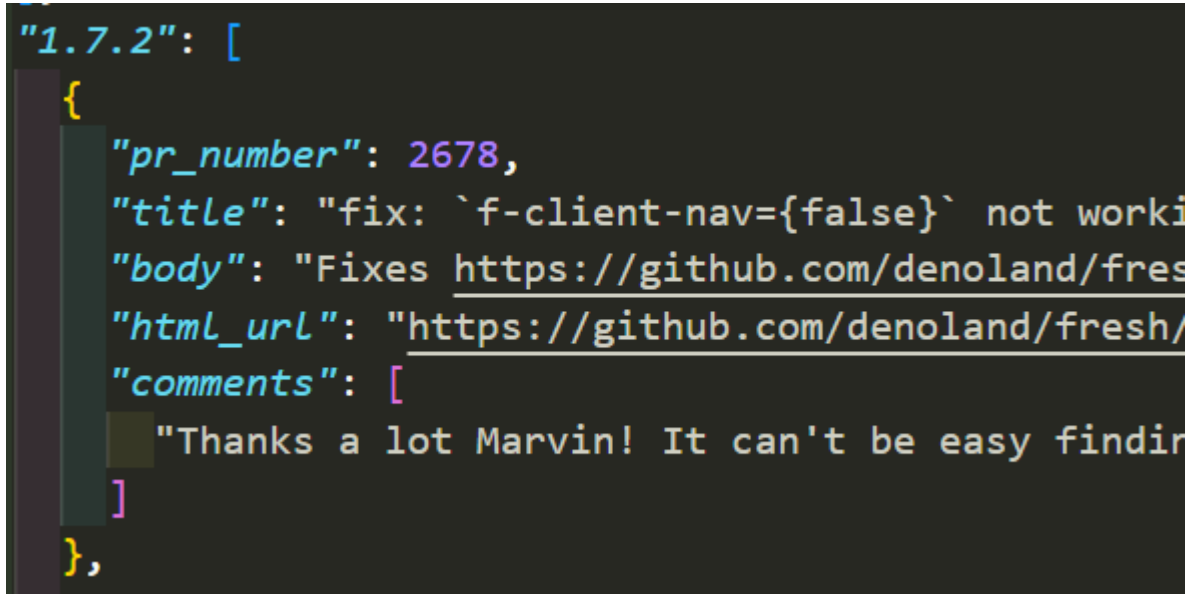




【実験方法】学習データの取得

GitHubリポジトリからGitHub APIを用いてデータを取得

- バージョン情報
- リリースノート
- バージョン毎のプルリクエスト



例: バージョン1.7.2に対応するプルリクエスト

【実験方法】
学習データとプロンプトを用いたアップデート

最初に与えるプロンプト

1. データの説明
   - プロダクト
   - 学習データ
2. 詳細な指示
   - 変更点が無い事も出力
   - 出力の根拠を明示 等
3. CursorでRAGを
   行うためのコマンド

**提案に合わせて以下の
ようなプロンプトを与える**

- ユーザーからフォローを入れる
- 提案の根拠を確認する指示

出力内容が堂々巡りする or
アップデート完了で終了

評価項目

RQ1（提案手法にてプロダクトのアップデートができるか）

1. アプリが実行可能か
2. 各モジュールの最新版に更新出来たか

RQ2（与えたデータはアップデートでどのように利用されるか）

1. 出力結果ではどのデータを参照しているか
2. 与えたデータを正確に解釈可能か

この項目に対して、**正常系（破壊的変更の影響を受けない）**と**異常系（破壊的変更の影響を受ける）**を設けてそれぞれ**5回ずつ**確認する。

異常系に実装した破壊的変更は以下の通りである。

- islandsディレクトリ外でHookを含めた実装をするとfreshアップデート後エラーになる
  → 意図的にroutesディレクトリでHookを使うように実装

結果（RQ1）

**提案手法にてプロダクトの
アップデートができるか**

プロダクトをアップデートする過程は以下のようになった。

正常系

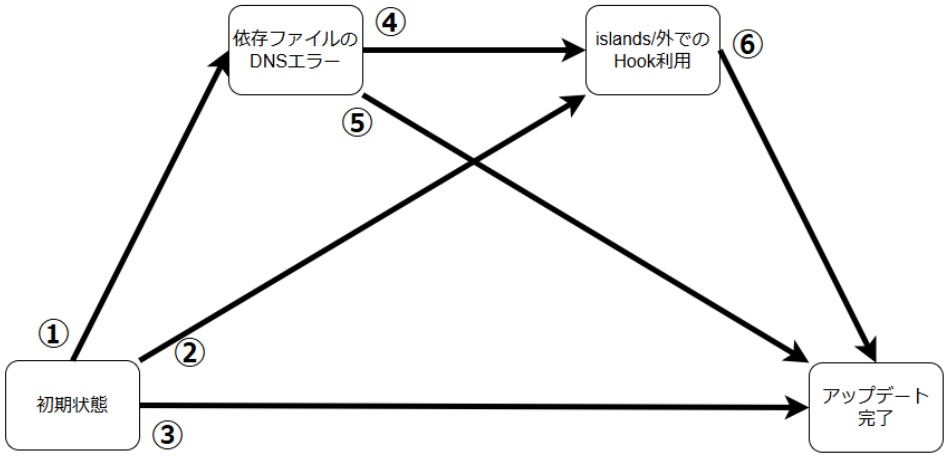
- 5回中4回でアップデートに成功
- 残りは「依存ファイルのDNSエラー」を
  解決できず失敗

DNSエラーについて

- 右図①の時にfreshの依存ファイルのURLが変更されたため、参照エラーが発生
  - Deno v1.39.1以降で解決

異常系

- 5回とも破壊的変更を乗り越えてアップデート完了



①： （共通） Deno(~v1.39.0) fresh(1.6.6~)

②： （異常系） Deno(v1.39.1~) fresh(1.6.6~)

③： （正常系） Deno(v1.39.1~) fresh(1.6.6~)

④： （異常系） Deno(v1.39.1~) fresh(1.6.6~)

⑤： （正常系） Deno(v1.39.1~) fresh(1.6.6~)

⑥： （異常系） Hookを islands/に移植

実験結果の概略図

結果（RQ1）

**提案手法にてプロダクトの
アップデートができるか**

各モジュールにおける最新版の提案結果は以下の通りである。

正常系

回数	fresh	Deno	preact	render-to-string
1	○	△(v1.37.2 → v2.1.2)	×(10.19.2)	○
2	○	○	更新なし	更新なし
3	○	△(v1.37.1 → v2.1.2)	×(10.22.0)	○
4	○	×	△(10.18.0 → 10.22.0)	更新なし
5	○	○	×(10.22.0)	○

異常系

回数	fresh	Deno	preact	render-to-string
1	○	△(v1.38.2 → v2.1.2)	×(10.22.0)	×(6.3.0)
2	○	×(v1.42.3)	×(10.22.0)	○
3	○	△(v1.37.2 → v2.1.2)	更新なし	更新なし
4	○	○	更新なし	更新なし
5	○	×(v1.45.3)	×(10.19.6)	更新なし

結果（RQ1）

**提案手法にてプロダクトの
アップデートができるか**

**いずれも場合においても成功したモジュールと
失敗したモジュールが見られた**

Fresh
- 全てのケースで成功

Deno
- DNSエラーに伴う修正策としてバージョンアップを
  行ったため成功したケースがあった

Preact
- 多くの場合で誤ったバージョン(10.22.0)を提案していた

**最新版
10.25.0**

Preact-render-to-string
- 更新されないことが多かった

結果（RQ2）

**与えたデータはアップデートで
どのように利用されるか**

出力結果でどのデータを参照しているか

最新版の探索

- PRのファイル
- バージョンリスト
- プロダクトで使用中のモジュールやバージョンを管理するファイル(import_map.json)

DNSエラー対応

- エラー文（DNSエラーが発生してインポートできない）
  （error trying to connect: dns error: failed to lookup address information: Name or service not known）

破壊的変更への対応

- freshのPRファイルとエラー文（Error: Hook "useState" cannot be used outside of an island component.）

結果（RQ2）

**与えたデータはアップデートで
どのように利用されるか**

与えたデータを正確に解釈しているか

最新版の探索　△

- PR内に記載の他バージョンを誤提案するケースが見られた
  （例: preactで10.22.0が提案）
  → 正確性に欠ける結果となった

DNSエラー対応　〇

- エラー文を解釈してNWの設定やcurlの実行、Denoアップデート等を提案
  → エラー内容の対応として適切だと思われる

破壊的変更への対応　〇

- エラー文を基に破壊的変更が行われたPR（feat: error if `useState` or `useReducer` is used outside of an island.）を適切に参照
  → 正確に解釈して対応できていた

得られた知見

(RQ1, RQ2)RAGを用いた提案はデータやプロンプトに精度が左右される

- 正しいバージョン提案ができていなかったケースがあった
  → Cursor（RAG）ではプロンプトと最も相性の高いデータの上位数件を参照することが原因か？
  → 検索対象のデータやプロンプトへの工夫が必要となるのでは

(RQ2)ユーザーのフィードバックが重要な場合がある

- ユーザーがNW設定を確認する等の実行結果を共有することで修正できたケースが見られた
  → DNSエラー等のコード外要因のエラーに対してはユーザーからのフィードバックが重要となるのでは

まとめ

- RAGとLLMを活用した依存関係のアップデートを本研究では提案した。アップデートについては概ね成功したものの、モジュールの最新版を正しく提案出来ないケースが見られた。

- RAGではプロンプトに最も関連した内容を優先的に取得するので、上手く提案できる場合とそうでない場合が見られた。

- 外部要因に絡んだエラーでは、ユーザー側のフィードバックが重要な場合がある。