

第 1 章

京大シラバス Agentic な時間割提案

1.1 はじめに

昨年度の 11 月祭では、我々は「完全一致のキーワード検索」にしか対応していなかった従来の公式シラバス検索システムに代わり、LLM による文章埋め込み（Embedding）技術を用いて、ユーザーの検索意図をより深く反映した抽象的な検索を可能にするシラバス RAG（Retrieval-Augmented Generation）システムの開発に取り組んだ [1, 2]。本年度は、2023 年以降に見られる Agent 型 AI の急速な発展を踏まえ、LLM を単なる情報検索の補助手段としてではなく、より能動的な「エージェント（Agent）」として活用することを目的とした。具体的には、ユーザーから提示される履修希望や興味・関心といった曖昧な要望を解釈し、時間割の空きコマや履修要件といった複雑な制約条件を考慮しながら、学期全体の時間割を自律的に構築・提案するシステムの開発を目指した。

1.2 背景技術：プロンプティング手法とエージェント構造

本プロジェクトで用いた LLM の制御手法として、基本となるプロンプティング手法である CoT[3] から、外部ツール連携の基礎となる ReAct[4]、そして Plan-and-Execute[5] のエージェント構造について順に解説する。

1.2.1 CoT (Chain of Thought)

Chain of Thought[3] は、大規模言語モデルに複雑な問題を解く際、最終的な答えに至るまでの推論過程を明示的に自然言語として生成させるプロンプティング手法である。単純な入出力のペアではなく、人間が問題を解く際の段階的な思考プロセスを模倣させることで、より正確で解釈可能な推論を実現する。単に「Let's think step by step（段階的に考えよう）」というフレーズを指示に加える（Zero-Shot-CoT）だけでも推論性能の向上が見られる。さらに、具体的な推論過程を例示する Few-Shot-CoT により、Fine-tuning などの追加学習を行うことなく、プロンプトの工夫のみで実装できる利点がある。

Standard Prompting

■ 質問: カフェテリアには 23 個のリンゴがありました。昼食に 20 個使い、さらに 6 個買いました。リンゴは何個ありますか？

回答: 答えは 27 個です。

Chain-of Thoughts Prompting

■ 質問: カフェテリアには 23 個のリンゴがありました。昼食に 20 個使い、さらに 6 個買いました。リンゴは何個ありますか？回答は順序立ててステップごとに出力してください。

回答: 最初に 23 個ありました。20 個使ったので、 $23-20=3$ 個残りました。そして 6 個買ったので、 $3+6=9$ 個になります。

CoT により、算術推論・常識推論・論理推論など様々なタスクでの性能改善が見られた [6]。さらに、CoT の性能改善はモデルの規模（スケール）に強く依存する「創発的性質（Emergent Abilities）」を示すことが知られている。一説には 100B（1000 億）パラメータを超えるような大規模モデルでは劇的な性能向上が観察される一方、それ未満のモデルでは標準的なプロンプティングと性能がほぼ同等に留まると報告されている。CoT によって LLM 内部の思考プロセスは改善され、推論能力は向上した。しかし、推論に用いる知識が LLM 内部に事前学習されたものに限られるという課題（知識の陳腐化やハルシネーション）は依然として残る。

1.2.2 ReAct (Reason and Act)

前述の CoT が扱った「推論（Reasoning）」能力と、外部環境に対する「行動（Action）」を結びつけたものが ReAct フレームワーク [4] である。ReAct は、LLM に「Reason（推論）」と「Act（行動）」を交互に生成させ、外部環境と相互作用しながらタスクを遂行するエージェントを実現する。ReAct における「ツール（Tools）」は、LLM と外部環境を接続する重要なインターフェースとして機能し、「Act（行動）」によって引数付きで呼び出される。ReAct フレームワークは、以下の中核要素が循環的に相互作用することで動作する。

(1) Thought（思考）

- 現在の状況と利用可能な情報を分析
- 目標達成に向けて次に必要な行動を計画
- 過去の観察結果を統合して推論を更新

(2) Action（行動）

- 特定のツールを選択し、適切なパラメータで呼び出し
- 外部環境への具体的な操作を実行
- 情報取得や状態変更を実現

(3) Observation（観察）

- ツールの実行結果を受信
- 新たに得られた情報を文脈に統合
- 次の思考プロセスへのインプットとして機能

ReAct は CoT と同様に、追加の Fine-tuning を必要とせず、few-shot prompting で実現される。

ReAct 応答例

■ 利用可能なツール:

Search[entity] Wikipedia API を通じてエンティティ情報を検索

Lookup[keyword] 現在参照中のページ内でキーワードを検索

Finish[answer] 最終回答を出力してタスクを終了

■ 質問: Apple Remote 以外で、Apple Remote が本来操作するように設計されていたプログラムを操作できる別のデバイスは何ですか？

■ 実行ログ (思考・行動・観察):

思考 1: Apple Remote を調べて、それがどのプログラムを操作するように設計されていたかを知る必要がある。

行動 1: Search[Apple Remote]

観察 1: Apple Remote は、2005 年に Apple から発売されたリモコンで、Front Row というメディアセンタープログラムを操作するために設計されていた。

思考 2: Apple Remote は Front Row を操作するために設計されていた。次に、他にどのデバイスが Front Row を操作できるかを調べる必要がある。

行動 2: Search[Front Row controls]

観察 2: Front Row は、Apple Remote または Mac のキーボードで操作できる。

思考 3: Front Row は Apple Remote と Mac のキーボードで操作できることが分かった。

答え: Finish[Mac のキーボード]

1.2.3 Plan-and-Execute

ReAct は強力なフレームワークである一方、いくつかの課題も存在する。ReAct は「思考」と「行動」を密接に繰り返す (Thought-Action-Observation のループ) ため、局所的な最適解に陥りやすい。つまり、タスク全体を見通した大局的な戦略の維持が困難となり、長期的な文脈理解が必要なタスクでは元の目標を見失ったり、非効率なステップを繰り返したりする場合があった。

Plan-and-Execute[5] は、これらの課題に対処するため、役割を二つの LLM (あるいは二つの役割) に分離するアプローチである。

Planner (計画者): Executor が使用可能なツール群を前提として、与えられたタスクを達成するための一連の行動計画 (サブタスクのリスト) を生成する。

Executor (実行者): Planner が生成した計画に基づき、各ステップを順次実行する。この際、ステップ内部では ReAct のように思考とツールの使用を (局所的に) 繰り返しながらサブタスクを遂行する。

効果と性能改善

Plan-and-Execute を採用する最大の利点は、認知負荷の分離にある。複雑・長期的なタスクにおいて、Planner は全体を俯瞰して計画を立案することに集中する。これにより、長期的な文脈の理解

や冗長な行動の回避といった、大局的な最適化が可能になる。一方、Executor は計画全体を意識する必要がなく、目の前の局所的なサブタスク実行に集中できる。

もう一つの利点として、解釈可能性と介入可能性の向上が挙げられる。ReAct が逐次的な思考の連鎖であるのに対し、本手法では最初に行動計画全体が明示されるため、システムが何をしようとしているのかを人間が容易に解釈できる。さらに、計画が生成された段階で人間がその内容をレビューし、修正を加えるといった介入も可能になる。

1.3 データ設計と前処理

京都大学が公開しているシラバスデータ [7] から、科目名・教員・成績評価・授業概要等の各種データを抽出し、検索に利用可能な状態へ前処理する基盤は、昨年度の RAG システム開発の時点で確立していた。

本プロジェクトにおけるデータ処理は、この昨年度のデータを基盤としつつ、従来の「人間による検索」用から「LLM がツールとして利用」しやすい形式への変更、および「時間割全体の最適化」に必要な情報の追加を主眼とした。

データの前処理に関する詳細（基本的な抽出ロジック等）は昨年度会誌の記載 [2] を参照されたい。本章では、昨年度からの主要な変更点と、エージェント化のために新たに追加した項目についてのみ詳述する。

1.3.1 シラバスデータの変更点

メタデータ

既存のシラバスデータから抽出したメタデータに対し、LLM が計算やフィルタリングに利用できるよう、以下の正規化と形式変更を行った。

- 単位数

従来「2 単位」のような文字列型 (str) であったデータを、エージェントが履修単位数を計算できるように数値型へ変更した。その際、「0.5 単位」といった例外が存在するため、整数 (int) 型ではなく浮動小数点数 (float) 型を採用した。

- 開講年度・開講期

「2025・前期」のように単一の文字列であったものを、「年度 (2025)」と「開講期 (前期)」の二つに分離し、メタデータ検索時の絞り込みを容易にした。

- 配当学年

「全回生」「2 回生以上」「主として 1・2 回生」など、表記揺れが多数存在したため、正規表現を用いてこれらを網羅的に解釈し、対象学年を示す数値のリスト (例: [1, 2]) に正規化した。(薬学部等、6 年制学部にも対応し 1~6 の範囲とした。)

シラバス本文

シラバス本文は、そのまま LLM に渡すには冗長であり、トークン長 (コンテキスト長) とコストを圧迫する。そこで、LLM の認知負荷低減と情報圧縮の観点から、用途に応じて二段階の要約レベルを生成した。

1. 検索結果・授業比較用の構造化要約

これは、検索ツールが LLM に提示するための要約である。LLM が時間割に未登録の授業を評価・比較するために参照するものであり、授業の主要な情報は保持しつつ冗長性を排除する必要がある。そこで、「授業のテーマ」「目的・背景」「主要トピック」といった項目をあらかじめ定義し、LLM に項目立てて構造化要約させた。これによりトークン数は平均で半分以下に抑えられた。また、副次的な効果として、情報の構造化による LLM の認知負荷軽減も期待される。

2. 現在時間割の確認用・短縮要約

これは、既に時間割に組み込まれている授業の情報を LLM に渡す際に用いる要約である。他の授業との比較などを行わないため、詳細な説明は不要であり、キーワードや単文レベルまで情報を圧縮した。

検索結果の要約例

この授業で学べること**中心テーマ:** 哲学史（古代から 18 世紀まで）* **目的・背景:** 哲学の基本的な問いを理解し、古典的テキストを通じて哲学史的知識を身につけること。* **主要トピック:** プラトン、アリストテレス、トマス・アクィナス、デカルト、ロック、カントの思想* **得られるスキル:** 哲学的な問いを自ら考える能力、哲学史に関する基礎知識
学習の進め方* **授業スタイル:** 古典テキストの読解を通じた講義形式* **学習の流れ:** 導入から始まり、各哲学者の古典を時系列で読み解く## 前提知識* **推奨される知識:** 特になし* **備考:** (授業内容からの推測)

現在時間割用の要約例

哲学史の基礎/古代から 18 世紀までの主要哲学者の思想/古典的テキストの読解/哲学の基本的な問いの理解

1.3.2 単位取得率データ

本プロジェクトでは、京都大学が公開している年度毎の「履修者数」と「単位取得者数」のデータ [8] を基に単位取得率を計算し、これを「単位取得の容易さ」を評価する指標の一つとして LLM に提供した。しかし、この単位取得率データには「科目名」と「開講部局」しか記載がなく、特に科目名は、一部が省略・変更されているため（例：「(演習)」やクラスコード「1S5」の欠落）、シラバスデータとの対応付けは困難であった。

今回は、特に楽単度合いの必要性が高いと想定される全学共通科目に絞り、以下の方法を用いてデータの紐付けを試みた。

1. 正規化: 双方のデータに対し、半角全角の統一、空白文字の削除を行う。
2. 前方一致検索: シラバスデータ側の授業名 S に対し、S.startswith(P) で完全に S に前方から含まれている単位取得率データ側の授業名 P を検索する。
3. 最長一致の採択: S に対して複数の P が前方一致した場合（例：S="哲学 II" に対し P1="哲学" と P2="哲学 II" が一致）、最も一致文字列が長い P（この例では P2）を採択する。
4. 開講部局による絞り込み: 上記で絞り込んでも複数の候補が残る場合、全学共通科目の開講部局である「国際高等教育院」のデータを優先する。
5. データの合算: それでもなお候補が残る場合（例：学部生用と大学院用の同一名称科目）、デー

タは区別せず、履修者数と単位取得者数を合算して扱った。

この手法により、全学共通科目 3154 個の授業のうち、約 95.6% にあたる 3015 個のデータ紐付けに成功した。一方、残りの 139 個は対応するものが見つからなかった。これら未対応データには、昨年度で廃講になった科目や、表記揺れの範疇を超えて名称変更された科目などが含まれると推測される。一部、名称が同一であるにも関わらず対応付けられなかったデータも存在し、これは今後の課題である。

1.3.3 データ処理における課題

現状のデータ前処理には、依然として多くの課題が残されている。

- シラバスデータの形式が学部・学科毎に異なっており、現状では特定の学部（例：医学部）に一切対応できていないほか、他学部でも一部データの抽出ミスが散見される。
- 履修要件、クラス指定科目、推奨科目といった重要な制約条件の自動抽出が実現できていない。
- 単位取得率データは、全学共通科目以外（専門科目）には未対応。専門科目は科目名の揺れがより大きく、対応付けが格段に困難になるためである。

1.4 時間割提案 Agent の基本構造

時間割提案 Agent の基本的な機能について説明する。

1.4.1 ユーザー情報の入力

ユーザーの情報や要望についての入力について、チェックボックスやスライダーを用いて確実に入力できる構造化データの項目と、時間割全体への要望や興味関心のある分野などテキストベースで入力する項目の2つを用意した。

確実に入力する構造化データ

- 学部・学科・学年
- 1限を許すか
- 空コマを許すか
- 全休を作りたいかどうか
- 授業を入れたくないコマ
- 授業形式への希望
- 授業評価方法の希望
- 既に履修することを決めている科目

テキストによる入力

テキスト入力欄を用い、主にユーザーの興味・関心分野や、上記項目では伝えきれない詳細な要望・制約を入力できるようにした。初期実装では単一のテキスト入力のみであったが、後に LLM を用いて対話的に要求を引き出すチャット形式に変更した。ただし、この対話による要求抽出プロセスの高度化は、本プロジェクトでは十分に改良できず、今後の課題として残されている。

1.4.2 LLM の利用ツール

エージェントが環境（シラバスデータベースや内部の時間割状態）と相互作用し、タスクを遂行するために、以下の 3 つのツールを定義し、LLM が ReAct 形式で呼び出せるよう設計した。

- 検索ツール

シラバスデータベースから授業を検索する。基本的には昨年度と同様、FAISS[9] を用いたシラバス本文のベクトル検索である。メタデータ検索では、複雑な and, or 検索まで可能ではあるが、初期実装では LLM のツール利用を簡素化するため、検索パラメータを「曜日・時限」と「検索クエリ」のみに限定した。Plan-and-Execute アーキテクチャへの移行後は、Planner の認知負荷が軽減されたため、任意パラメータとして「開講部局」「分野」「配当学年」等も指定可能とし、より高度な絞り込みを実現した。検索結果を ToolMessage の形式で、「授業 ID」「授業名」「曜日・時限」「授業形式」「評価方法」「単位取得率」「シラバス要約」を JSON 形式で返す。該当する検索結果が見つからなかった場合は、ToolMessage で検索結果が見つからなかったことを返す。

- 授業追加ツール

指定した ID の授業を現在の時間割に追加する。授業名による指定は、LLM の出力（表記揺れ）によって不安定となり、状態管理を複雑にするため、一意な授業 ID を指定する形式を採用した。ToolMessage を用いて、授業の追加が成功したか否かを返す。失敗した場合は、その理由（「月曜 2 限が重複」等）も併せて返す。

- 終了ツール

時間割提案完了・計画の各ステップのタスクの完了を判断した際に呼び出し、次のステップへと移行する。

1.4.3 LLM へのデータ入力形式

エージェントの思考ステップにおいて、プロンプトに含める現在の状態として、以下の情報を渡す。具体的には以下のような形式で渡す。

```

# 現在の状態
## 現在の時間割情報
- **現在の時間割**:
月曜 1 限: 1227
月曜 2 限: 610
月曜 3 限: 無し
〜〜〜
金曜 5 限: 無し
- **集中講義**: 無し
- **各授業 id の詳細**:
1227: '科目名': '英語リーディング ER30', '担当教員': '非常勤講師', '授業形式': ['演習'],
'評価': '平常点': 60, '課題': 0, '発表': 0, '討論': 0, '小レポート': 0, '小テスト': 0, '期末
レポート': 40, '期末試験': 0, 'シラバス本文': "アカデミックリーディングの基礎/学術入門
書の精読/英語の学術的語彙/近代自然科学の歴史に関する知見"
- **総取得単位数**: 21.0
- **学部**: 理学部
- **学科**: 理学科
- **現在の学年**: 1
## ユーザーからの要望
- 一時限目には授業を入れないでください
- 一日の内で授業と授業の間に空いている時間があっても良い
- 授業が一つも入っていない曜日が作りたい
- 授業形式は、講義、演習が好ましく、実験は避けたいです。
- 授業の評価方法は、講義、演習が好ましく、実験は避けたいです。
- **避けたい曜日・時限**: 木曜 5 限
**興味のある内容**: "1. **学びたい分野**: - 数学系（特に応用的な分野） - 金融や保険数
学 2. **興味のある応用先**: - 金融 - 保険数学 - その他の数学の応用先（具体的には未記載）
3. **人文社会系の科目**: - 経済 - 法律 - 社会に出てからも活躍しそうな内容. **授業形式
の希望**: - 授業にあまり出たくない - 期末レポートや期末テストの割合が高い形式を希望"

```

1.5 ReAct 型 Agent

本プロジェクトの初期実装として、LLM が環境と相互作用しながら自律的にタスクを遂行する ReAct アーキテクチャ [4] を採用した。将来的な一般公開時の運用コストを考慮し、LLM モデルには低コストな「gpt-4o-mini」を使用した。

1.5.1 初期実装と課題

初期の設計では、検索・授業追加・終了の 3 つのツールのみを用い、単一の LLM が「現在の状態（時間割情報）」と「ユーザー要望」に基づき、「思考」と「行動」を逐次的に繰り返す、シンプルな ReAct エージェントを構築した。

しかし、この構成で検証を行った結果、実用上、以下の深刻な問題点が多発した。

- 探索の非効率性
関連する授業が見つからない場合、LLM は同一または類似の検索クエリを際限なく発行し続け、検索の無限ループに陥る。
- 状態認識の欠如
既に授業追加済みのコマを「空き」とであると誤認し、重複して授業追加を試み、失敗する。エージェントはこの失敗から学習できず、同じコマへの追加を繰り返した。
- タスク完了判断不能
時間割が十分に埋まった、あるいはこれ以上追加すべき授業がないという「タスク完了」の条件を LLM 自ら判断できず、設定した最大ステップ数に達して強制終了するケースが多発した。
- トークン数制限
ReAct は設計上、過去の「思考・行動・観察」の全履歴をコンテキストに含める。これに加えて「現在の時間割詳細」という静的な情報も毎回のプロンプトに含めるため、思考が長くなると、API のトークン数上限に容易に到達した。

1.5.2 課題への対策

上記の問題点を解決するために、次の 2 点を変更した。

一つ目は、プロンプトによる「空コマ」の明示である。時間割の情報から空いているコマを判断させてしまうと LLM の負荷が上がってしまうため、事前に空コマを求め明示的に与えることとした。これにより、既に埋まっている時間に授業を追加してしまう事だけでなく、検索の段階で埋まっている時間を検索してしまうという問題まで、大幅に削減できた。

二つ目は、思考履歴のトークン数による削減である。全履歴を渡す代わりに、トークン数で履歴を打ち切り、最新の思考のみをコンテキストに含めるよう変更した。この変更後も、エージェントのタスク遂行性能に顕著な低下は見られず、コストと API 制限の問題を実用的なレベルで解決した。

1.5.3 ReAct 型の限界

前節で述べた対策では根本的な問題は解決せず、システムプロンプトによって検索・授業追加・完了判断の思考方法について指示をしても、検索の失敗、不毛な授業追加の繰り返し、そしてタスク完了の判断ミスといった問題を本質的には解決できなかった。また、ユーザーの興味関心のある授業を検索・追加することはできるが質が高いとは言えず、一般的な履修の決め方・履修要件などはほとんど考慮して考えることが出来なかった。検証のため、gpt-5-mini など少し上位のモデルを使用したところ、検索と授業追加に顕著な性能差が見られた。これによりタスクの成否が LLM の推論能力に強く依存することは確認できた。しかし、これにより我々は、本プロジェクトにおける最大のジレンマ、すなわち「性能とコストのトレードオフ」に直面することとなった。

低コストモデルでは、履修要件やユーザーの要望まで理解して検索・授業追加をすることが出来ず、高性能モデルでは要件を理解することはできるが ReAct 型の思考と行動の繰り返しにより、API 料金が嵩んでしまう。この「性能」と「コスト」のトレードオフは、ReAct 型アーキテクチャを本タスクで実用化する上での根本的な障壁であると結論づけた。この問題を解決するため、我々はアーキテクチャそのものを見直し、「複雑な思考」と「単純な実行」を分離する必要があると考えた。これが、

次章で述べる Plan-and-Execute アーキテクチャへの移行の直接的な動機である。

1.6 Plan-and-Execute 型 Agent

前章で述べた ReAct 型の根本的な課題、すなわち「性能」と「コスト」のトレードオフを克服するため、我々は Plan-and-Execute (計画・実行) アーキテクチャ [5] へと移行した。本アーキテクチャは、その構造的特性から、このジレンマに対する直接的な解決策となると判断したからである。

第一の利点は、LLM が担うべき認知的な負荷を分離することが出来る点にある。

- Planner は、履修要件、ユーザーの曖昧な要望、時間割全体のバランスといった、大局的で複雑なコンテキストの理解に集中する。
- Executor は、複雑な全体要件を考慮する必要がなく、Planner から指示された個別のサブタスクの実行という、単純化されたタスクに集中できる。

第二の利点は、この役割分離により、役割に応じた LLM モデルの使い分けが可能となる点である。これは、ReAct 型が直面した最大のジレンマに対する直接的な解決策となる。

- Planner は、複雑な要件を正確に解釈し、質の高い実行計画を立案するため、高性能・高コストな LLM を割り当てる。計画の立案はタスクの初回に一度だけ実行されるため、API コストへの影響は最小限に抑えられる。
- Executor は、Planner の計画に従ってツールを実行する (比較的単純な) 役割であるため、低コストな LLM を割り当てる。Executor は「検索」や「追加」のたびに何度も呼び出されるが、安価なモデルを用いることで、全体の API コストを劇的に低減できる。

1.6.1 Plan-and-Executor 型の実装

Planner

Planner は、「ユーザー情報」「履修要件」「確定済みの授業」「推奨科目リスト」「履修戦略のヒント」といった情報をマークダウン形式で受け取り、実行計画を立案する。

特に、Planner が複雑な思考に集中できる利点を活かし、LLM が単独では知り得ない情報 (ドメイン知識) を参照情報としてプロンプトに組み込んだ。これは Few-Shot プロンプティングのように機能し、より質の高い計画を生成するよう出力を誘導した。

出力は、Executor が局所的な判断に陥らないよう、全体の方針を示す「全体の指針」と、具体的なサブタスクのリストである「計画」を含む厳格な JSON 形式として定義した。

Executor

基本は今までの ReAct 型を原型としたが、役割に合わせて以下の点を変更した。

- システムプロンプトで、全体の時間割を自律的に考えるための指示を削除し、Planner の計画に従って検索・授業追加を行うように指示
- 従来の ReAct 型では、現在の時間割に加えユーザー情報などを毎回渡していた。しかし、Executor はユーザー情報を解釈する必要はないため、プロンプトから時間割以外の冗長な情報を削除した。各ステップの実行に必要な情報は、すべて Planner が生成した計画から渡される。

1.6.2 改善点

ReAct 型から Plan-and-Execute へ変更したことにより、生成される時間割の質が大幅に改善された。

第一に、本構成の主目的であった Executor の動作安定性が向上した。ReAct 型では Executor 自身が思考する必要があったため、無駄な検索や失敗が多発していた。一方、本構成では Executor の役割が「計画の忠実な実行」に単純化された。Planner から「検索クエリの方向性」や「授業の比較基準」が明確に与えられるため、よりユーザーの要望に沿った授業選択が可能となった。

第二に、時間割全体の質が向上した。Planner が高性能モデルの推論能力を活かし、質の高い計画を立案できた場合に限定されるものの、その計画を Executor が実行することで、人間が考案したものと遜色のないまともな時間割を提案できるケースも確認された。しかし、この改善は、システム全体のボトルネックが「Executor の実行能力」から「Planner の計画立案能力」へとシフトしたことを意味する。LLM が本質的に知り得ない、各学部・学科固有の「ヒューリスティックな履修戦略」を Planner に理解させ、質の高い計画を安定して生成させることは極めて困難であり、本稿執筆時点でもプロンプトの試行錯誤が続いている。

1.7 今後の課題と展望

本システムは現段階では未完成であり、多くの課題が残されている。

- データ処理について、履修要件や推奨科目などまだまだ対応しきれていないものが多く残っている。
- 学部学科ごとに異なるようなヒューリスティックな履修戦略について、一部学部にしか対応できていない。
- LLM のコンテキスト理解とトークン数のバランスを取った時間割情報のどの渡し方の試行錯誤。
- 計画の質が良ければ上手く時間割を考えられるが、Planner の質の良い計画立案の制御がまだ完全にはできていない。
- Executor がタスクを完了できなかった際に、対応しきれず次のステップに移ってしまう。Executor で問題が起こった場合に、Planner へ戻り全体を見通して計画の修正ができるようにしていく。
- 時間割を一度提案するだけでなく、提案した時間割をベースとしてユーザーとの対話しながら修正・改善できるようにする。

今回の 11 月祭までには、一般に公開できるレベルまでシステムの完成度を高めることはできなかった。しかし、来春の新生が入学する時期を目標とし、本章で挙げた課題の改善と試行錯誤を継続する。最終的には、より高精度かつ多機能な時間割提案エージェントとして公開できるよう、引き続き開発を行っていく。

参考文献

- [1] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.
- [2] KaiRA. 京大シラバス検索 rag システム, 2024.
- [3] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.
- [4] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023.
- [5] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models, 2023.
- [6] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023.
- [7] 京都大学. Kulasis-シラバス一覧. https://www.k.kyoto-u.ac.jp/external/open_syllabus/all, 2025. Accessed: 2025-11-10.
- [8] 京都大学. 各授業の平均履修登録者数. <https://www.kyoto-u.ac.jp/sites/default/files/inline-files/10-1-fd74b48ede8655ef7a87d79175a75c51.pdf>, 2024. Accessed: 2025-11-10.
- [9] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus, 2017.