2025 年度

修　士　論　文

大規模言語モデルを用いた
ガイド付推論連鎖型複合サービス推薦

指導教員: 村上 陽平

立命館大学大学院 情報理工学研究科
情報理工学専攻 博士課程前期課程
計算機科学コース

学生証番号: 6611240056-8

氏名: 松本 賢司

大規模言語モデルを用いたガイド付推論連鎖型複合サービス推薦

松本 賢司

内容梗概

複数の API を連携させて作成したサービスのことを複合サービスという．複合サービスにより，個々に提供される API よりも機能が拡張されたサービスを提供することができる．しかし，複合サービスの設計において，自然言語で記述されるユーザ要求は抽象的かつ不完全な記述の場合が多々ある．そのため，有用な API を的確に発見し，選択することが困難という課題がある．従来の API 推薦では，要件文を数値ベクトルとして表現し，API の説明文のベクトルとの類似度や，過去の複合サービスにおける API の利用傾向を基に，適切な API を推定することが一般的であった．しかしながら，これらの手法は，推論結果がどのような判断過程に基づくものかを把握しにくいという課題がある．こうした中で，大規模言語モデル(以下，LLM)の登場により，推論手法に新たな展開が生じている．LLM は大規模コーパスを用いた事前学習を通じて高い文脈理解および推論能力を獲得しており，要件文中の曖昧さや省略された情報を文脈から補完可能になる．しかしながら，Chain-of-Thought や Few-Shot といったプロンプト形式は，一般性を重視してドメイン非依存となるように設計されているため，専門的なタスクに適合した推論プロセスとなる保証がない．

そこで，本研究では，LLM に推論手順とその手順に準拠した事例を与え，その手順通りに要件文から API を推薦する手法を提案する．具体的には，ソフトウェア工学におけるトップダウン型の機能分解に基づき，要件文に含まれる目的や機能を階層的に整理する推論手順を設計する．LLM がこの推論手順に従って段階的に推論を行うガイド付推論連鎖により，要件文と API の対応付けを行う．本手法の実現にあたり，取り組むべき課題は以下の 2 点である．

LLM の推論過程を導く手順の設計

要件文は抽象的であり，API は膨大であるため，要件文から直接 API を推薦することは困難である．そのため，要件文から機能を抽出し，それを API に対応付けるための推論手順を明示的に定義する必要がある．

定義した推論手順に基づいた事例の作成

推論手順を提示するだけでは，判断の粒度や推論の流れが LLM に十分に共有されず，推論の省略や逸脱が生じる可能性がある．そのため，設計した推論手順に基づく推論事例を提示する必要がある．

一つ目の課題に対しては，LLM の推論を段階的に誘導するための推論手順を設計した．具体的には，要件文から主要機能を抽出し，それに対応する API カテゴリを選択，各カテゴリに所属する API から候補を列挙し，最終的に API を決定する四つの段階からなる推論の流れを定義した．これにより，LLM は各段階に対応した推論を順に実行できる．推論の逸脱や不整合な判断を抑え，目的から API の選択までを一貫した流れとして進めることが可能になる．

二つ目の課題に対しては，過去の複合サービスで使用されている API，その所属カテゴリ，および要件文を固定し，それらと整合するように，LLM に基づく補完推論を用いて事例を作成した．具体的には，カテゴリ選択の段階では，使用されている API の所属カテゴリのみを出力した．次に，カテゴリ内の API の列挙段階では，正解 API に加えて同一カテゴリに属する複数の API を含める構成にした．最終的な API の推薦では，候補の中から使用されている API のみを出力するようにした．これにより，実際の API 推薦において最終的な推薦に至るまでに複数の候補が存在し得る推論事例を作成することができた．

これらを踏まえ，ProgrammableWeb に掲載されている複合サービス 1497 件の実データから，提案手法によるカテゴリおよび API の推薦結果を，Precision，Recall，F1 を用いて提案手法の有効性を検証した．さらに，特定の LLM の性能に依存していないかを確認するため，GPT，GPT-oss，LLaMA，DeepSeek を用いて精度比較を行った．また，既存プロンプト手法との比較に加え，提案手法の推論手順や事例を部分的に取り除いたアブレーション実験を行い，各要素が推薦の性能に寄与していることを確認した．本研究の貢献は以下のとおりである．

LLM の推論を導く手順の設計

要件文から主要な機能を抽出し，カテゴリの選択，API の複数提示，最終決定の推論手順を明示的に定義した．これにより既存手法に比べて F1 のカテゴリ推薦の精度が約 2%，API 推薦の精度が約 1.3%向上した．また，アブレーション実験により提案手法の有用性を示した．

定義した推論手順に基づいた事例の作成

過去の複合サービスで使用している API とカテゴリおよび要件文から，中間の推論過程を補完することで，事例を生成した．この推論事例のみをプロンプト内に提示したアブレーション実験の結果，既存の事例を用いた設定と比較して，カテゴリ推薦および API 推薦の精度がそれぞれ 7%，10%向上した．

Guided Reasoning Chains for API Recommendation

Kenji Matsumoto

Abstract

A composite service integrates multiple APIs to provide more features than a single API. However, natural-language requirements for composite service design are often abstract and incomplete. Therefore, developers struggle to find and choose useful APIs. Traditional API recommenders turn requirements into vectors. They choose APIs by comparing API descriptions and past mashup usage, but people cannot easily explain their choices. LLMs learn context from training on large text. They can fill in missing or unclear details in requirements. However, prompts such as Chain-of-Thought and Few-Shot do not ensure the right reasoning for this task. Therefore, we propose a method that gives LLMs a clear reasoning process and examples to recommend APIs from requirements. We design a reasoning process to break requirements into goals and functions. We use Guided Reasoning Chains to map these requirements to APIs step by step. This study addresses the following two challenges.

Designing Reasoning Procedures to Guide LLMs

Requirement texts remain abstract, and APIs exist in large numbers. Therefore, directly recommending APIs from requirement texts is difficult. Therefore, this study must explicitly define a reasoning process to extract functions from requirement texts and align them with APIs.

Generating examples based on the defined reasoning process

This method only shows the reasoning steps and does not fully share the decision granularity or reasoning flow with LLMs. Because of this limitation, LLMs may omit steps or deviate from the intended reasoning process. Therefore, this study presents reasoning examples based on the designed reasoning process.

To address the first issue, this study designs a reasoning process to guide LLMs through a step-by-step reasoning procedure. Specifically, this study defines a four-step reasoning process: extract core functions from requirement texts, select API categories, list candidate APIs in each category, and choose the final APIs. This design enables LLMs to execute step-by-step reasoning and avoid reasoning

drift and inconsistent decisions.

To address the second issue, this study fixes APIs, categories, and requirement texts from past composite services and creates consistent reasoning examples using LLM-based reasoning interpolation. In the category selection stage, the model outputs only the used API category. In the subsequent API selection stage, the model includes the correct API and several other APIs from the same category. In the final step, the model outputs only the API selected from the candidates. Because of this design, this method creates reasoning examples to allow multiple API candidates before the final API selection.

Based on these points, this study evaluates the proposed method on 1,497 real composite services from ProgrammableWeb to measure category and API recommendation accuracy using Precision, Recall, and F1. We compared accuracy using GPT, GPT-oss, LLaMA, and DeepSeek to check whether the results depend on a specific LLM. In addition, this study conducts ablation experiments to partially remove the proposed reasoning process and reasoning examples, along with comparisons to existing prompt methods. These experiments verify that each component contributes to recommendation performance. The contributions of this study are as follows.

Designing Reasoning Procedures to Guide LLMs

This study explicitly defines a reasoning process to extract core functions from requirement texts, select categories, list multiple APIs, and choose final APIs. Compared with existing methods, the proposed method improves the F1 score by up to 2% for category recommendation and by 1.3% for API recommendation. In addition, ablation tests demonstrate the usefulness of the proposed method.

Generating examples based on the defined reasoning process

This study generates examples by interpolating intermediate reasoning processes from APIs, categories, and requirement statements used in past composite services. An ablation study that presents only these reasoning examples in the prompt improves category and API recommendation accuracy by 7% and 10%, respectively, compared with settings that use existing examples.

大規模言語モデルを用いたガイド付推論連鎖型複合サービス推薦

目次

第 1 章 はじめに **1**

第 2 章 API 推薦 **3**

2.1 API 推薦……4

2.2 LLM を用いた推薦……5

第 3 章 ガイド付推論連鎖型プロンプト **7**

3.1 汎用プロンプト……7

3.1.1 Zero-Shot……7

3.1.2 Chain-of-Thought ……8

3.1.3 Few-Shot ……9

3.1.4 Few-Shot-CoT ……10

3.1.5 Plan and Solve……11

3.2 ガイド付推論連鎖プロンプトの概要 ……12

3.2.1 基本構成……12

3.2.2 設計手順 ……15

3.3 推論手順の定義 ……16

3.4 推論手順に準拠した事例の作成方法 ……16

3.4.1 補完推論の必要性 ……16

3.4.2 事例作成プロンプト ……17

第 4 章 API 推薦への適用 **20**

4.1 API 推薦における推論手順…… 20

4.1.1 トップダウン設計……21

4.1.2 トップダウン設計に基づく推論手順 ……21

4.2 API 推薦における事例生成プロンプト ……24

4.3 API 推薦におけるガイド付推論連鎖型プロンプト ……28

第 5 章 評価 **30**

5.1 評価手法 …… 30

5.1.1 実験データ …… 30

5.1.2 比較手法……32
5.1.3 利用したモデル……34
5.2 結果……36
5.3 アブレーションテスト……38

第 6 章 考察 **40**
6.1 ハルシネーション分析……40
6.1.1 手法間比較による発生頻度……40
6.1.2 1 要件文内での発生規模……41
6.2 事例選定による精度変化……41
6.2.1 事例選定戦略と精度への影響……42
6.2.2 実例に基づく分析……43
6.3 推論に失敗した複合サービスの分析……44
6.3.1 推論段階別に見た失敗ケース……44
6.3.2 全手法が推論に失敗した複合サービス……46
6.4 要件文が推薦精度に与える影響……46

第 7 章 まとめ **48**

謝辞 **50**

参考文献 **51**

付録 **53**
A.1 ハルシネーションの詳細……53

第1章 はじめに

近年，公開 API の急増に伴い，サービスコンピューティングの重要性が高まっている．特に，複数の API を連携させて構築される複合サービスは，単独の API では実現困難な機能やユースケースに対応できる手段として注目されており，サービスの柔軟性と開発効率の向上に大きく寄与している．例えば，地図情報を提供する API と天気情報を共有する API を連携させることで，地図上に各地の天気情報を重ね合わせて表示する複合サービスを容易に構築できる．

しかし，複合サービスの設計には，膨大な数の API の中から目的に適した API を発見し，組み合わせることが依然として難しいと言う課題がある．このような API 推薦の困難さの一因が，ユーザ要件である．自然言語で記述される要件文には，必要な機能が明示的に含まれてない場合が多く，抽象的な表現にとどまることが少なくない[1]．そのため，要件文から必要な機能を適切に汲み取る必要がある．この問題に対し，BERT を用いて要件文と API 説明文から意味ベクトルを学習し，要件に合う候補 API を絞り込んだ後，各候補と要件文の関連度を評価して推薦する手法がある[2]．しかしながら，この手法では推薦の根拠が十分に説明されないという課題がある．こうした背景から，近年では大規模言語モデル(以下，LLM)を API 推薦に応用する試みが進んでいる．LLM は大規模なコーパスを用いた事前学習により，高度な文脈理解能力を獲得しており，要件文に含まれる曖昧な表現や省略された情報を文脈から推論することが可能になる．この特性は，自然言語の要件文から推薦するタスクにおいて，有望な手法となり得る．しかしながら，Chain-of-Thought(以下，CoT)や Few-Shot といった従来のプロンプト手法は汎用性を重視して設計されているため，API 推薦のような専門的なタスクに適合した推論プロセスを実行できるとは限らない[3][4]．

そこで本研究では，ソフトウェア工学におけるトップダウン型の機能分解に基づき，要件文から階層的に推論を行う手法を提案する．具体的には，API 推薦のドメインに特化した推論手順を定義する．さらに，この推論手順に従って推論が行われるよう，ガイド付推論連鎖型プロンプトを設計する．設計したプロンプトの事例に推論手順を組み込むことで，LLM が一貫した推論プロセスを辿るように誘導する．本手法を実現するにあたり，取り組むべき課題は以下の 2 点である．

LLM の推論を導く手順の設計

要件文から API を推論するためには，LLM が一貫した推論を形成できるように，ソフトウェア工学に基づいた推論構造を設計する必要がある．

定義した推論手順に基づいた事例の作成

設計した推論手順に基づく推論事例を提示し，LLM が一貫した推論過程を模倣できるようにする必要がある．

本論文の構成は次のとおりである．第 2 章では関連研究について述べる．第 3 章では，ガイド付推論連鎖型プロンプトについて述べる．第 4 章では API 推薦へのガイド付推論連鎖プロンプトの適用について詳述する．第 5 章で実験結果を示し，第 6 章でそれらに対する考察を行う．最後に第 8 章において，今後の展望および課題について述べ，本論文の結論とする．

第2章 API 推薦

既存の API 推薦手法について説明する．複合サービスはその構成要素となる API を適切に選択できるかどうかが，サービスの品質や開発効率に大きく影響する．複合サービスの構築事例として図 1 の Uber Eats[1] の開発事例を用いる．Uber Eats とはユーザがレストランから食品をオンラインで注文し，その食品を配達するサービスである．このサービスはユーザが食品の注文を確定する際に，決済を行う API である Stripe が利用される．Stripe を通じてユーザは簡単に多数の支払い方法を選択することができる．支払い完了後，ユーザは地図と位置情報の API である Google Map を利用して，店舗の位置も含め，注文した商品の位置情報と到着時間をリアルタイムで追跡することが可能になる．このように複合サービスを構築する際には，要求される機能を満たす API を適切に発見し，選択する必要がある．しかしながら，オンライン上に存在する膨大な数の API の中から，ユーザが求める機能を持つ API を発見することは非常に困難である．API の利用に関する統計より，最大 85.6%の API が複合サービスに利用されていないことからも，多くの有用な API が見過ごされている可能性が高い[5]．このような背景から，ユーザ要件に基づいて適切な API を自動的に推薦する研究が活発に行われてきた．本章では，これまで提案されてきた従来型のサービス推薦手法と LLM を用いた推薦システムを取り上げ，それぞれの特徴と課題について説明する．

図 1: 複合サービスの例

[1]https://www.ubereats.com/jp

2.1 API 推薦

API 推薦は，複合サービスの機能的なニーズを満たす適切な API 群を特定することを目的とするタスクであり，多くの手法が提案されている．複合サービスは複数の API を組み合わせて構成されるため，要件文の内容だけでなく，API 間の関係性を考慮した推薦が求められる．このような背景から，API 推薦に関する研究では，着目する情報やモデル構造の異なる様々なアプローチが検討されてきた．要件文や API の利用履歴といった複数の情報を組み合わせて推薦を行う手法として，Wu らは MTFM を提案した[6]．MTFM は要件文の特徴と利用履歴に基づく相互作用特徴を融合して推薦し，さらに Mashup カテゴリ予測を補助タスクとして同時学習することで，その学習信号を推薦に反映して精度向上を狙う．次に，API 間の関係性に着目した手法として，Liu らは，API 間の共起関係とその時間的変化を動的なグラフとして扱い，グラフニューラルネットワークを用いて API の特徴を学習する DySR を提案した[7]．DySR では，要件文と API をそれぞれ異なる特徴量として学習した後，それらを対応付けるための変換を導入することで，要求と API の関係を直接比較可能にしている．この仕組みにより，利用履歴が少ない API に対しても推薦が可能となっている．次に，API に関する知識を事前学習によって獲得する手法として，Wang らは，ServiceBERT を提案した[8]．ServiceBERT は，API の説明文，操作名，入力と出力パラメータなどのテキスト情報を用いて，Transformer ベースのモデルを事前学習する手法である．事前学習では，API タグ付けタスクを通じて，各 API が持つ機能の特徴や API 間の関連性を学習させている．事前学習済みのモデルは，API のタグ付けや複合サービスにおける API 推薦といった下流タスクに適用され，API の説明文に基づいた推薦を可能としている．その結果，ServiceBERT は高い推薦性能を示している．

これらの API 推薦手法は，要求文の内容，API の利用履歴，API 間の関係性，および API に関するテキスト情報など，それぞれ異なる情報に着目している点に特徴がある．一方で，これらの手法は，要求文に必要な機能や目的が比較的明示的に含まれていることを前提としており，自然言語で記述される抽象的な要求に対しては十分に対応できない場合がある．また，最終的な推薦結果のみが提示され，推薦に至る判断過程や理由が明示されないことが多く，説明性の観点で課題が残されている．

2.2 LLM を用いた推薦

近年，LLM の発展に伴い，自然言語で記述された要件文を直接入力として用い，推薦を行う手法が提案されている．従来の API 推薦手法では，要件文を特徴量に変換した上で推薦を行うものが主流であった．これに対し，LLM を用いた手法は，推論や生成能力を直接活用できる点に特徴がある．これらの手法は，要件に含まれる文脈情報や暗黙的な意図を考慮できる可能性があることから，近年注目を集めている．Gu らは，複数の処理ステップから構成されるサービス実行手順を推薦する問題を対象として，Plan, Generate and Match(以下，PGM)と呼ばれる三段階の枠組みを提案した[9]．PGM は 3 段階で構成され，まず要件文からタスクの解決手順を Plan として生成し，次にその計画に沿って要件文から構造化された仕様を Generate し，最後に生成した計画と仕様を用いて候補ワークフローとの一致度を計算して Match する．Plan 生成は複数の事例を LLM に与える Few-Shot により行い，計画データの大規模な追加学習を必要としない点を特徴としている．このように，推薦前に「要件文から段階的な手順へ落とし込む工程」を明示的に置くことで，複雑な要求の理解と推薦の一貫性を高めている．一方で，API 推薦そのものを LLM の生成問題として定式化し，ドメイン適応を行う研究もある．Qin らは，API 推薦に対して LLMAR を提案し，LLM を用いて要件文から API 名の列を生成する生成型の推薦として扱う手法を提案した[10]．LLMAR では，API 推薦に必要な知識を LLM に取り込むために，API の説明文の生成やカテゴリ付与，複合サービスのカテゴリ推定など，複数の関連タスクをインストラクション形式で学習させる設計をしている．さらに，学習を一度に行うのではなく，多段階で段階的に微調整する方針を取り，計算資源を抑えるために LoRA による微調整を行う．このような設計は，単なるプロンプトの利用に比べて，API 領域の知識や利用パターンを学習により取り込み，推薦精度を改善することを目的としている．さらに，LLM を情報の抽出器として用い，グラフなどの構造情報と組み合わせて推薦性能を高める方向性も報告されている．Feng らの LLMSARec は，API の説明文を LLM で解析して API の情報を推定して生成し，その情報を用いて API の特徴を強化する[11]．加えて，同一の複合サービスに対応する API 同士は関連が強く，異なる複合サービスに対応する API 同士は関連が弱くなるように学習させることで，API 間の関係に基づく情報と API の説明文に基づく情報を整合させている．このように，LLM 単体で完結させるのではなく，説明文から得られる意味と履歴に基づいたグラフから得られ

る関係性を統合することで推薦精度の向上を図っている．

これらの LLM を用いた推薦手法は，要件文に含まれる文脈情報や暗黙的な意図を考慮できることから，従来の特徴設計に基づく手法と比較して，高い推薦性能が報告されている．一方で，推論過程がモデル内部の処理に大きく依存している．その結果，推薦に至る判断が汎用的な推論になりやすく，対象タスクに固有の推論過程や制約を明示的に反映することが難しいという課題がある．

本研究では，対象タスクに特化した推論手順を LLM に与え，推論手順に準拠した事例を与える．これにより，抽象的な要件文に対しても，対象タスクに基づいた意味理解が可能になる．また対象タスクに特化した推論が可能になる．

第3章 ガイド付推論連鎖型プロンプト

LLM は，入力として与えられる文章の構成（以下，プロンプト）に応じて出力を生成するため，同一のモデルであっても，プロンプトの設計によって得られる結果が大きく異なるという特徴がある．特に，推論を伴う要件文では，推論の進め方や判断の粒度が明示されていない場合，推論過程の省略や逸脱が生じ，出力が不安定になることがある．このような特性を踏まえると，LLM から望ましい出力を安定して得るためには，要件文の意図や制約，および推論の進め方を適切に与える入力設計が重要となる．そのため，近年では，LLM の挙動を制御するための入力設計手法であるプロンプトエンジニアリングが注目されている．

本章では，まず 3.1 節において，代表的なプロンプト設計手法について説明する．次に 3.2 節では，本研究で提案するガイド付推論連鎖型プロンプトの概要を示す．さらに，3.3 節ではガイド付推論連鎖型プロンプトで用いる推論手順の作成方法について述べ，3.4 節では推論手順に準拠した事例の作成方法について説明する．

3.1 汎用プロンプト

LLM は，事前学習に加えて，事後学習を通じて人間の指示に従う能力を獲得している．事後学習では，人手で作成された指示と回答のペアを用いた教師あり微調整や，人間のフィードバックに基づく強化学習が適用される．これにより，LLM は自然言語による指示を解釈し，それに応じた出力を生成することが可能となる．さらに，事後学習を経た LLM は，文脈内学習と呼ばれる特性を示す．文脈内学習とは，再学習を伴うことなく，入力プロンプト内に要件文の指示や少数の事例を含めるだけで，モデルがその要件文に適応する能力である．この特性により，LLM の挙動は，入力として与えられるプロンプトの構成や記述内容に大きく依存する．

以上のことから，LLM の性能を引き出すためには，モデルの事後学習によって獲得された特性を踏まえたプロンプト設計が重要となる．本節では，このような特性を活用した一般的な汎用プロンプト設計手法について説明する．

3.1.1 Zero-Shot

Zero-Shot とは，図 2 に示す通り，自然言語による指示文と対象となる要件文のみを与え，それに基づいて回答を出力するプロンプト設計手法である[12]．こ

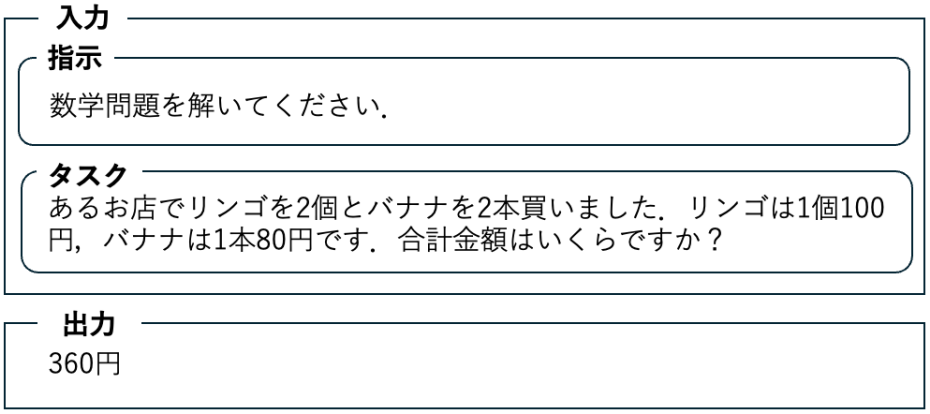

図 2: Zero-Shot の例

の手法では，LLM が事前学習により獲得した知識を利用して指示に従った応答を行う．プロンプト設計が容易であり，要件文の内容が比較的単純で，出力結果のフォーマットに制約を必要としない場合に適している．しかし，プロンプトに与える情報が限られているため，期待する出力と異なる結果が生成される場合がある．また，多数の処理を必要とする要件では Zero-Shot の適用が困難である．さらに，モデルの事前学習に出力が依存しやすいと言う課題もある．

3.1.2 Chain-of-Thought

CoT とは，3.1.1 節で述べた Zero-Shot に”Let’s think step by step”定型文を追加する手法である．本手法の適用例を図 3 に示す．Zero-Shot は要件文からの結果のみ出力するのに対し，CoT では，モデルに推論課程を段階的かつ明示的に生成させることを目的としている．しかし，推論の形式や粒度が明示的に規定されていないため，生成される思考過程の一貫性が不安定になりやすいという課題がある．具体的に，推論の途中で誤った過程や不要な要素が挿入され，最終出力の正確性に悪影響を及ぼす場合がある．

入力

指示

数学問題を解いてください．

タスク

あるお店でリンゴを2個とバナナを2本買いました．リンゴは1個100円，バナナは1本80円です．合計金額はいくらですか？

定型文

Let’s think step by step.

出力

リンゴは1個100円なので，2個で 100円 × 2 = 200円
バナナは1本80円なので，2本で 80円 × 2 = 160円
合計は 200円 + 160円 = 360円

図 3: Chain-of-Thought の例

3.1.3 Few-Shot

Few-Shot とは，少数の入出力事例をプロンプト内に提示し，その対応関係を

入力

指示

数学問題を解いてください．

事例

Q. あるお店でパンを3個と牛乳を1本買いました．パンは1個150円，牛乳は1本200円です．合計金額はいくらですか?

A. 650円

⋮

タスク

あるお店でリンゴを2個とバナナを2本買いました．リンゴは1個100円，バナナは1本80円です．合計金額はいくらですか？

出力

リンゴは1個100円なので，2個で 100円 × 2 = 200円
バナナは1本80円なので，2本で 80円 × 2 = 160円
合計は 200円 + 160円 = 360円

図 4: Few-shot の例

学習させることでモデルに回答を生成させるプロンプト設計手法である．本手法の適用例を図 4 に示す．Few-Shot では，各事例において入力に対する最終的な出力のみが提示され，推論過程は明示されない点に特徴がある．このような構成により，モデルは事例間の入出力パターンを抽象化し，類似の入力に対して適切な出力を生成しやすくなる．特に，出力形式や文体，階層構造構造などを揃えたい場合に有効であり，Zero-Shot と比較して出力の安定性が向上する傾向がある．一方で，推論過程が提示されないため，モデルがどのような判断基準や内部推論に基づいて結論を導いたかは制御できない．その結果，表面的なパターン模倣に依存した回答が生成されやすく，複雑な論理構造や多段階の判断を要する要件においては，誤った推論に基づく出力が生じる可能性がある．

3.1.4 Few-Shot-CoT

Few-Shot-CoT とは，Few-Shot をベースに少数の推論過程を含めた入出力事例をプロンプト内に提示し，その形式に倣ってモデルに推論させて回答を生成させるプロンプト設計手法である．本手法の適用例を図 5 に示す．本手法では，各事例の出力において， 特徴がある．これにより，事例の推論形式にモデルが適用するため，推論の粒度と構造が揃いやすく出力が安定しやすい特徴がある．

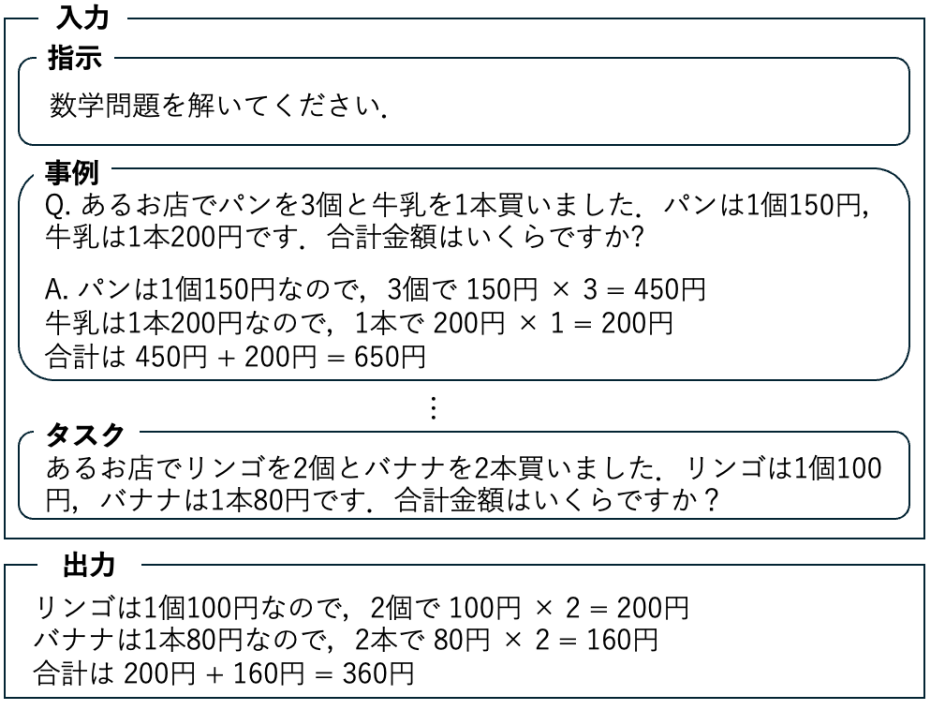

図 5: Few-Shot-CoT の例

しかし，推論形式を模倣できる一方で，各手順の条件や目的が明示的に定義されないため，モデルは適用範囲を暗黙的に推定して推論を行う．この結果，タスク固有の制約や判断基準が十分に考慮されないまま推論が進行し，誤った前提に基づく結論や，不適切な推論過程が生成される場合がある．

3.1.5 Plan and Solve

Plan and Solve とは，3.1.1 節で述べた Zero-Shot プロンプティングに” Let’s first understand the problem and devise a plan to solve the problem. Then, let’s carry out the plan and solve the problem step by step.”という定型文を追加する手法である[13]．本手法の適用例を図 6 に示す．Plan and Solve では，モデルにタスクを解くためのプランを生成し，それに従って解いていくことを目的としている．これにより，推論の順序が整理されるため，Zero-Shot や CoT に比べ，計算タスクの時では，計算エラーが少なくなるという特徴がある．しかし，プランはモデルが自由に決めるため，生成されるプランが同じタスクでも毎回異なることが起きるという課題がある．また，生成されるプラン内に不要なステップや重要なステップが欠落する場合がある．

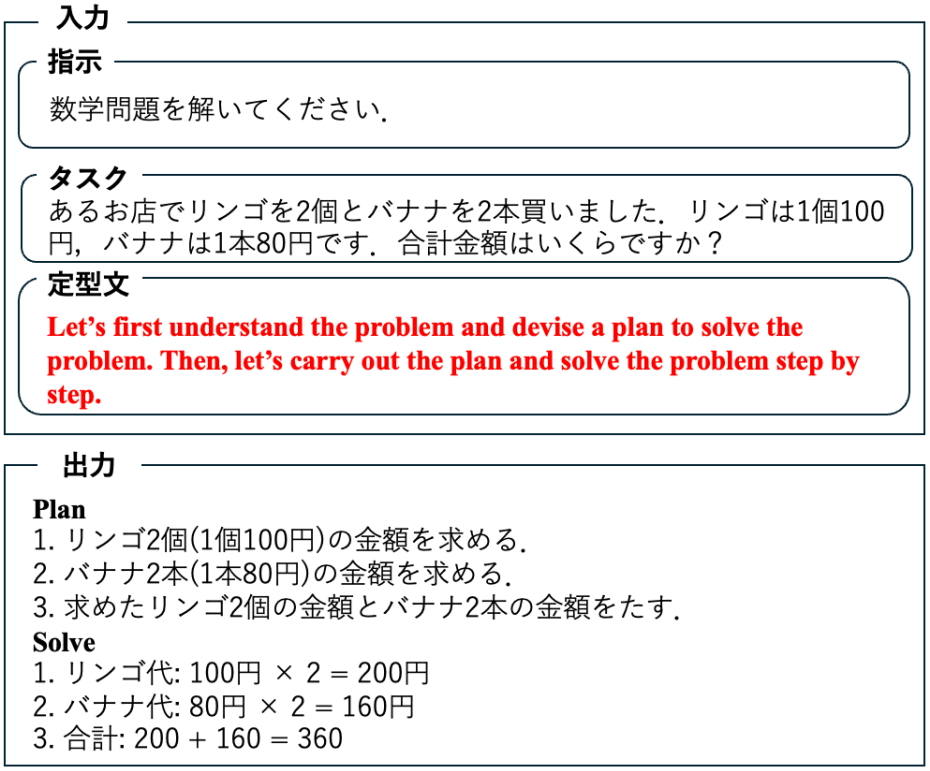

図 6: Plan and Solve の例

3.2 ガイド付推論連鎖プロンプトの概要

3.1 節では，代表的な汎用プロンプト設計手法について説明した．これらの手法は，多様なタスクに対して一定の性能向上をもたらす一方で，特定のタスクに対して推論過程や判断基準を明示的に制御することは難しいという共通の課題を有している．特に，複数の判断を段階的に行う推論タスクにおいては，推論手順の省略や順序の逸脱が生じやすく，その結果として誤った結論が導かれる場合がある．

本研究では，このような課題に対処するため，特定のドメインにおける多段階推論タスクを対象として，推論の進め方そのものを明示的に制御し，誘導するプロンプト設計手法である，ガイド付推論連鎖型プロンプトを提案する．本プロンプトは，あらかじめ推論手順を明示的に定義し，その手順に準拠した事例をプロンプト内に組み込むことで，LLM が設計者の意図した推論過程を一貫して辿るように誘導する点に特徴がある．

本節では，まずガイド付推論連鎖型プロンプトの基本構成について説明した後，本プロンプトを作成するまでの設計工程の全体像を示す．

3.2.1 基本構成

ガイド付推論連鎖型プロンプトは，前節で述べた Few-Shot-CoT と Plan and Solve の考え方を基礎として作成したものである．その構成図を図 7 に示す．プロンプトは「指示」,「推論手順」,「推論手順に沿った事例」,「推論に参照する情報」,「入力」,「出力形式」の 6 つの要素で構成される．以下の 2 つを主軸としてプロンプトを構築する．

・推論手順の明示的な定義
・推論手順に準拠した事例の提示

1 つ目について，本手法では，タスクを完了するために必要なステップを事前に定義し，これを推論手順としてプロンプトに明記する．これにより，モデルの自由な推論プロセスの生成を抑制し，作成者が意図した特定の手順に沿って問題を解くように機能する．推論手順の作成方法については 3.3 節で述べる．

2 つ目について，LLM に与える Few-Shot において，推論手順に準拠した推論過程を含む事例を作成し，プロンプトに埋め込む．これにより，モデルに対して入力からどのように推論を展開して出力に至るかという構造的な例を与えることができる．

結果として，未知の入力データに対して，モデルは事例と同じ推論手順を辿

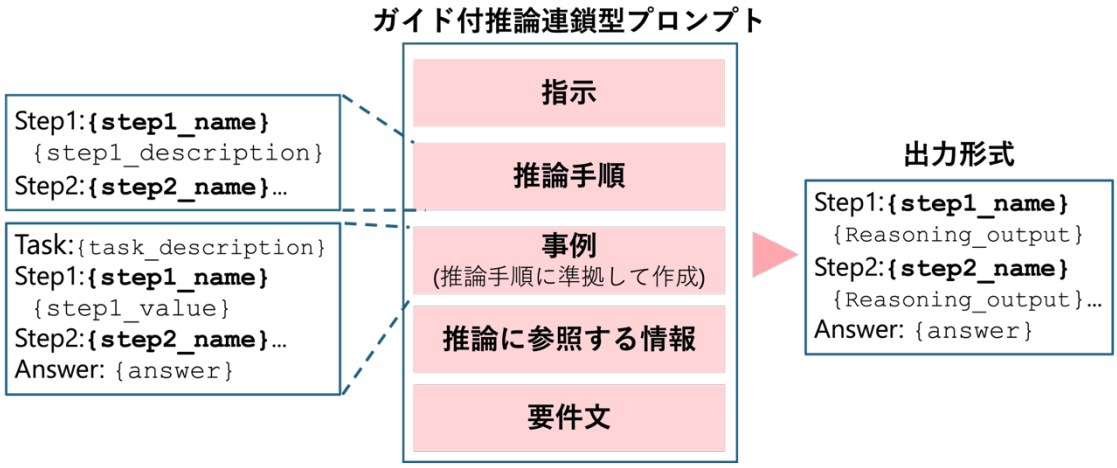

図 7: タスク特化型プロンプトの構成図

り，粒度を模倣して回答を生成することが可能となる．具体的な事例作成方法については 3.4 節で述べる．

以上の構成により，LLM の推論過程を暗黙的な挙動に委ねるのではなく，設計者が定義した手順とその実例に基づいて制御することを可能とする．また，中間的な思考過程が明示されるため，各ステップにおける判断根拠が可視化され，結果に対する説明可能性も向上する．図 8 に具体的なプロンプト例を示す．なお，ガイド付推論連鎖型プロンプトの設計枠組みを示すため，特定のドメインごとに具体化される箇所についてはフォントを変更して表現している．本プロンプトは，以下の 6 つの要素から構成される．

1. **指示**

   LLM に対して，本タスクにおける役割と目的を明確に定義する要素である．{Target_name} には LLM に出力して欲しい名称を入力し，{Domain_name}には何の[Requirements]なのかを説明する単語を入力する必要がある．

2. **推論手順**

   タスクを遂行するためにモデルが辿るべきステップを定義する要素である．思考の手順を明示的に記述することで，モデルの推論方法を固定化する．{StepN_process}は各推論手順の名称を入力する必要がある．

3. **事例**

   ある要件文と，2.で提示した推論手順に準拠した具体的な出力の組を提示する要素である．これによりモデルに要件文からの解法のフォーマットを学習

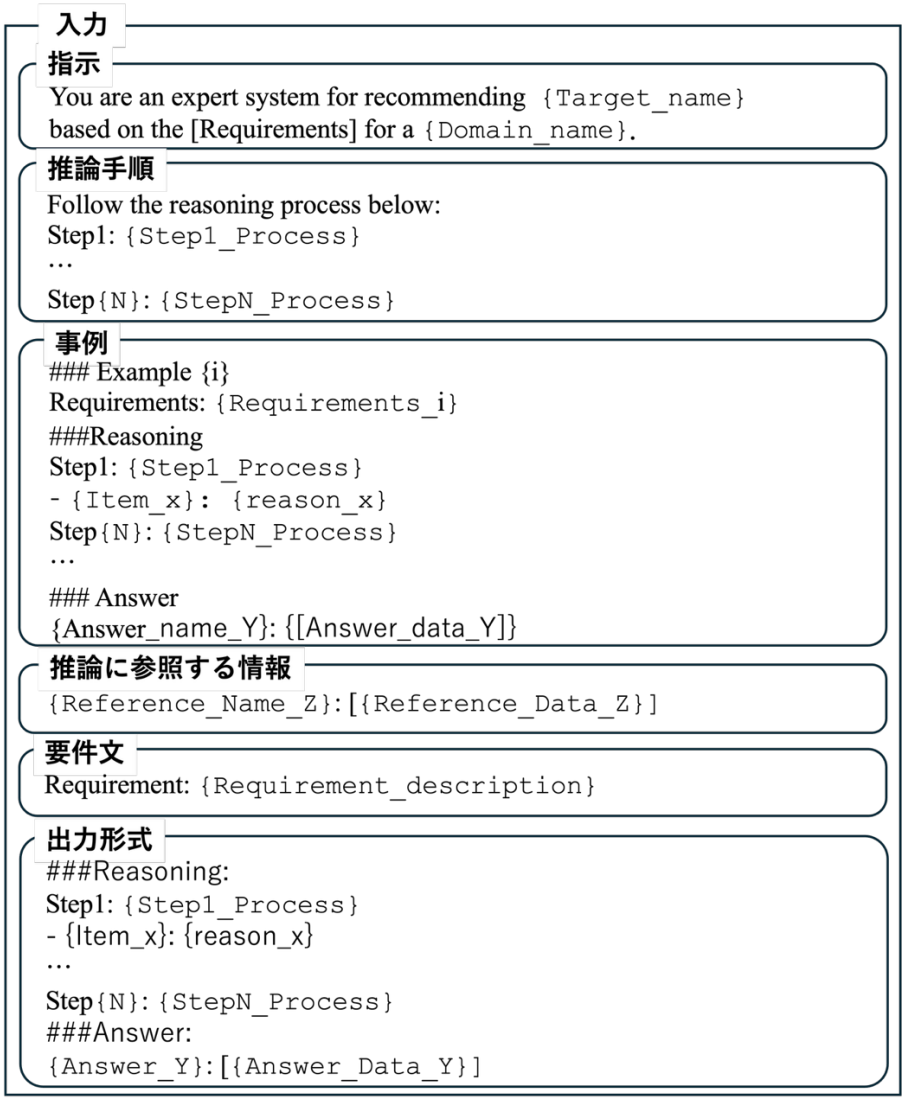

図 8: ガイド付推論連鎖型プロンプトの例

させる. なお, 事例の具体的な構成や選定方法については, 3.4 節で述べる. {Requirements_i} には各事例に用いた要件文を入力する. {StepN_process}は推論手順で定義した各推論手順の名称と対応付して入力する必要がある. {Item_x}は推論されたものを入力し, {Reason_x}にはその推論理由を入力する必要がある. {Answer_name_Y}は出力したい要素の総称を入力し, {Anser_data_Y}は LLM から出力された要素が入る.

4. 推論に参照する情報

LLM の推論中に参照可能な補助情報を与える要素である. ここで与える情

報は，構造化された形式で提示する必要がある．{Reference_Name}には参照する要素の総称を，{Reference_data}は参照する要素が入る．

5. 要件文

LLM に与えるタスクを{Requirement_description}に配置する．

6. 出力形式

モデルが生成する回答のフォーマットを指定する要素である．この出力形式は事例と同じ形式にする必要がある．

このプロンプトにより，モデルは与えられたタスクに対して，どのような順序で推論を進めるべきかを事前に把握した状態で回答生成を行うことが可能となる．さらに，タスク入力とは独立に「推論に参照する情報」を提示している点に特徴がある．この構成は，複数の選択肢の中から推論する必要があるタスクにおいて有効である．出力形式においては，推論手順で定義された各ステップに対力応する形で出力構造を指定している．

以上の構成により，ガイド付推論連鎖型プロンプトは，多段階の推論を要するタスクにおいて，出力の安定性と再現性を向上させる効果が期待できる．

3.2.2 設計手順

ガイド付推論連鎖型プロンプトは，汎用プロンプトのように即座に適用できるものではなく，設計工程を伴うため，その一連の設計手順を図 9 に示す．はじめに，対象とするドメインを分析し，そのドメインにおけるタスク達成に必要な推論手順を定義する必要がある．最終的な出力を得るまでに要求される判断や処理を定義することで，ドメイン固有の制約条件や判断基準を推論過程に反映させることが可能となる．次に，前段で定義した推論手順に準拠した事例を作成する．ガイド付推論連鎖型プロンプトでは，事例は定義された推論手順と厳密に対応する必要があるため，人手による作成は容易ではない．そこで，本研究では，推論手順に従った事例を生成するための専用プロンプトを別途設計し，事例作成に用いた．事例作成プロンプトの詳細については 3.4 節で述べる．最後に，推論手順および事例をもとに，ガイド付推論連鎖型プロンプト全体を構成する．

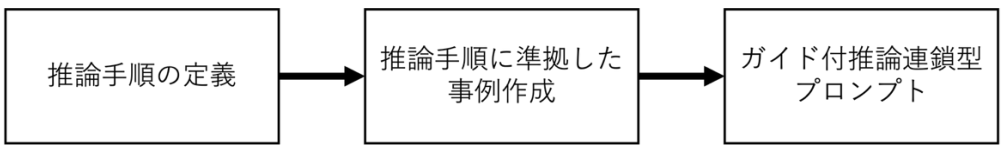

図 9: ガイド付推論連鎖型プロンプトの設計手順

以上の設計手順により，ガイド付推論連鎖型プロンプトは，特定のドメインにおける要求に対して，多段階推論の一貫性を確保し，出力の安定性を向上させることが可能となる．

3.3 推論手順の定義

本節では，ガイド付推論連鎖型プロンプトにおいて用いる推論手順を，設計対象としてどのように定義するかについて述べる．推論手順は，単なる推論の流れを記述したものではなく，ドメインに特化した判断構造を反映した設計要素として位置付けられる．推論手順の設計においては，推論を複数のステップに分割し，各ステップが担う判断内容を明確に定義する必要がある．また，ステップ間に依存関係がある場合，関係性を明示的に定義する必要がある．さらに，各ステップにおいて参照すべき情報を明示することが重要となる．参照すべき情報が存在する場合には，その情報名を角括弧（[]）で囲んで記述することで，自然言語による本文ではなく，参照情報であることを明示する．これにより，処理したい要件文は参照情報を基に，LLM が推論手順に沿って推論することが可能になる．本研究では，推論時に参照すべき情報を 3.2 節で説明したガイド付特化型プロンプト内の「推論に参照する情報」の箇所で提示する．

以上の設計方針に基づき，本研究では，ドメインに特化した多段階推論タスクを対象として推論手順を定義する．次節では，これらの推論手順に準拠した事例をどのように作成するかについて述べる．

3.4 推論手順に準拠した事例の作成方法

事例を与えるプロンプトでは，事例に含まれる推論過程の構造が，未知の入力に対する推論結果に大きな影響を与える．

本研究では，LLM を用いて定義した推論手順に厳密に従った推論過程を含む事例を作成し，これをプロンプト内に提示する．本節では，推論手順に準拠した事例の作成に必要となる推論の内容と，そのプロンプト設計手法について説明する．

3.4.1 補完推論の必要性

一般的なデータには，要件文と最終的な判断結果，および判断対象に関する付加的な情報のみが与えられることが多く，定義した推論手順に対応する推論過程が明示的に与えられるとは限らない．このため，推論手順の一部に対応する

情報は存在する一方で，中間的な判断を導出する過程や，各段階において複数の候補を比較しながら検討する過程は明示されない．また，各推論ステップにおいて，なぜその判断に至ったのかという理由情報も与えられない場合が多い．そのため，推論手順に準拠した事例を作成するためには，既存データに含まれない中間的な推論過程および選定理由を既存データから補完する必要がある．しかし，これらの推論過程をすべて人手で作成することは，多大な時間とコストを要する．

そこで，本研究では，LLM を用いて，要件文および関連する入力情報を与えることで，定義した推論手順に沿った中間的な推論過程を補完的に生成する補完推論を提案する．

3.4.2 事例作成プロンプト

図 10 に，推論手順に準拠した事例を生成するために用いるプロンプトの全体構造を示す．本プロンプトは，図 8 と類似した構造を持ち，推論手順および出力

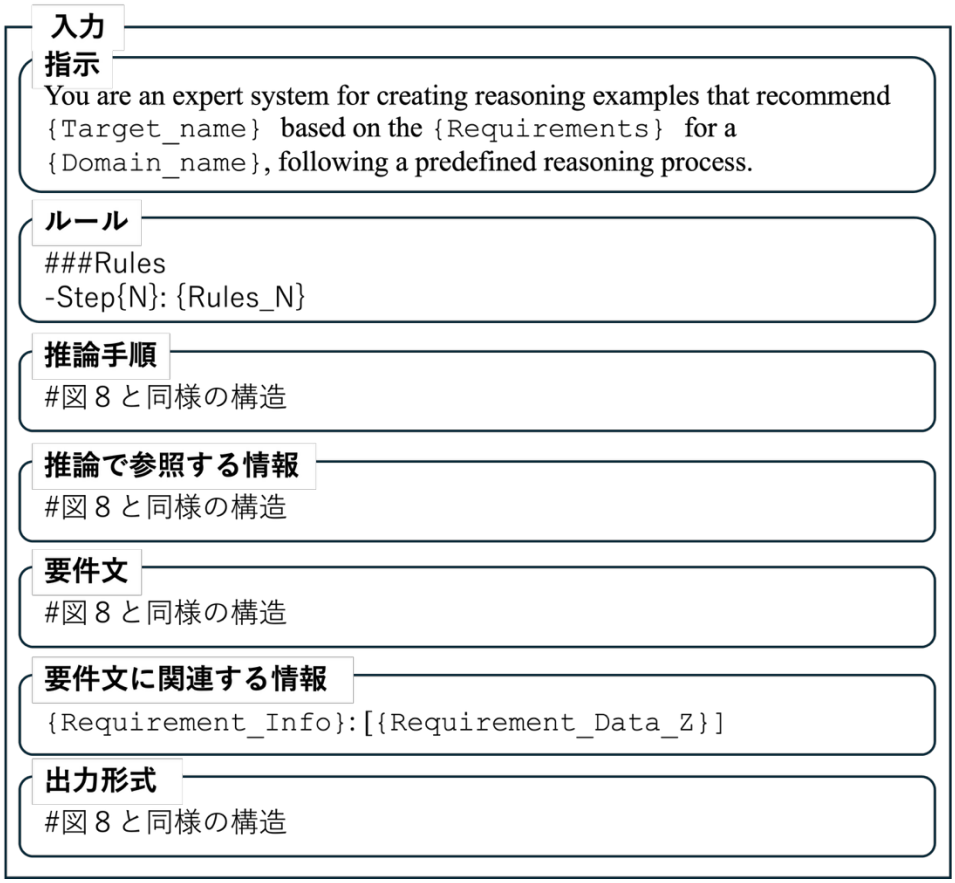

図 10: 推論手順に準拠する事例作成プロンプト

形式を一致させることで，提示される事例と実際の推論時の振る舞いとの間に構造的な整合性を持たせている．これにより，モデルが事例中の推論過程をそのまま模倣しやすくなり，定義した推論手順に沿った推論を安定して行えるようになる．本プロンプトは，以下の 7 つの要素から構成される．

1. **指示**

   要件文に基づいて適切な推薦結果を導出するタスクに対し，推論手順に準拠した事例を生成する役割を定義する．本要素では，モデルに対して事例生成を行う主体としての振る舞いを明示する．`{Target_name}`には LLM に出力して欲しい名称を入力し，`{Domain_name}`には何の[Requirements]なのかを説明する単語を入力する必要がある．

2. **ルール**

   事例生成時に遵守すべき制約を明示する要素である．ここでは，推論手順の各ステップにおいて出力すべき内容を規定する．これにより，正解として与えられている出力情報を保持したまま，中間的な推論過程のみを補完的に生成することが可能となる．

3. **推論手順**

   3.2 節で述べたガイド付推論連鎖型プロンプトと同一の推論手順を用いる．これにより，事例生成時と推論時とで，推論の流れおよび判断の粒度が一致し，Few-Shot 学習における模倣効果を高めることができる．

4. **推論で参照する情報**

   推論過程において参照可能な補助情報を与える要素である．これらの情報は，判断対象の種類や候補要素に関する記述を，構造化された形式で提示する．モデルはこれらの情報を参照することで，補完推論を行いながら推論過程を生成する．

5. **要件文**

   推論手順に準拠した事例を生成したいタスクの要件文を記述する．

6. **要件文に関連する情報**

   要件文に対応する最終的な判断結果および，それに付随する属性情報を与える要素である．この情報は，「推論で参照する情報」と同一の形式で記述され，推論過程において正解となる判断結果を明示する役割を持つ．この要素を与えることで，補完推論によって生成される中間的な推論過程が，正解から逸脱しないよう制御される．`{Requirement_Info}`は要件文の何に関連

するかの名称を入力し，{Requirement_Data}はその要素を入力する．

7. 出力形式

ガイド付推論連鎖型プロンプトで用いるプロンプトと同一の出力形式を採用する．

以上の構成に基づき，本プロンプトの出力と，事例作成プロンプトで入力した要件文を組み合わせることで，推論手順に準拠した事例が作成可能となる．この事例と推論手順をガイド付推論連鎖型プロンプトに組み込むことで，ドメインに特化した推論が可能になる．

第4章 API 推薦への適用

3 章で述べたガイド付推論連鎖化型プロンプトを API 推薦に応用する方法について述べる．本章では，4.1 節にて API における推論手順について述べる．4.2 節において，API 推薦における事例作成プロンプトについて述べ，4.3 節に API 推薦におけるガイド付推論連鎖型プロンプトについて述べる．

4.1 API 推薦における推論手順

複合サービスにおける API 推薦タスクの課題として, 利用可能な API の数が膨大である点が挙げられる．公開されている API はそれぞれ異なる機能や用途を持つ一方で，同一または類似した機能を提供する API も多数存在している．そのため，要件文から直接 API を選定しようとすると，同時に考慮すべき API の候補が非常に多くなり，判断の負担が大きくなる．特に，自然言語で記述された要件文は抽象的であることが多く，どの API が要件に適合しているかを即座に判断することは容易ではない．この結果，推薦の過程が複雑化し，適切な API を選定することが困難になる．さらに，複合サービスでは，複数の API を組み合わせてサービス全体を構築する必要がある．この場合，各 API が個別に要件を満たしているかだけでなく, API 同士を組み合わせるかという点についても考慮する必要がある．すなわち，API 間の役割分担や機能の補完関係を踏まえた上で, サービス全体として整合性の取れた API 構成を選定することが求められる．このような観点は，単一 API の推薦に比べて，複合サービスにおける API 推薦をより困難なものにしている．

このような背景から，API 推薦タスクでは，要件文に含まれる目的や必要機能をもとに，それらを段階的に具体化しながら，最終的な API の組み合わせを提案する推論手順が重要となる．要件文を直接 API に対応付けるのではなく，途中段階で必要な機能を整理，分解し，判断の粒度を徐々に細かくしていくことで，推薦の一貫性や妥当性を確保することが可能となる．

本節では，このような課題に対処するため，API 推薦タスクにおける推論手順について述べる．特に，本研究では，自然言語で記述された要件文から API を選定するまでの過程を明示的な手順として定義し，モデルが一貫した判断過程を辿れるようにすることを目的とする．

4.1.1 トップダウン設計

ソフトウェア工学における設計工程では，トップダウン設計に基づき，システム全体の目的や要求を起点として，機能や構成要素を段階的に詳細化していく手法が一般的に用いられる．ソフトウェア工学では，複雑なソフトウェアシステムを効率的かつ高品質に開発するために，要求分析，設計，実装，テスト，保守といった工程からなる開発プロセスが一般的に用いられる．特に，上流工程である要求分析や設計の内容は，後続の実装やテスト工程に大きな影響を与えるため，初期段階での検討が重要となる．これらの工程のうち，要求分析の工程では，システムが満たすべき要求や目的を明確化することが求められる．これに対して，設計工程では，要求分析によって整理された要求をもとに，システムを構成する要素とそれぞれの役割，および要素間の関係性を定義する．設計工程の重要な役割は，抽象的に記述された要求を，実装可能な具体的構成へと段階的に落とし込むことにある．この工程が不十分である場合，要求と実装との対応関係が不明確になり，結果としてシステム全体の品質低下を招く可能性がある．大きな機能単位からより詳細な機能単位へと段階的に分解し，分解された各機能に対応する具体的な構成要素や実装単位を定義することで，要求と実装との対応関係を明確にする．このように，目的から手段へと段階的に詳細化していく点が，トップダウン設計の特徴である．API 推薦タスクにおいても，要件文はシステム全体の目的を自然言語で表現したものであり，その内容は抽象的であることが多い．そのため，要件文を直接 API に対応付けようとすると，要求の解釈が曖昧になり，適切な API 選定が困難になる場合がある．この問題を回避するためには，要件文に含まれる目的や必要な機能を整理して分解し，それぞれに対応する具体的な API 構成を段階的に決定する必要がある．

以上の理由から，本研究では，ソフトウェア工学におけるトップダウン設計の考え方を API 推薦タスクに適用する．これにより，要件文から API 選定に至るまでの判断過程を明確化し，一貫した推論手順に基づいて API 推薦を行うことを目指す．

4.1.2 トップダウン設計に基づく推論手順

本研究では，要件文の解釈から API 選定に至るまでの過程を，LLM に対して段階的に導くため，明示的な推論手順を定義する．この推論手順は，前節で述べたトップダウン設計に基づく機能分解の考え方を採用しており，要件文に含まれる抽象的な目的や意図を，段階的に具体化しながら API 選定へと結び付ける

ことを目的としている．これにより，要件文が抽象的である場合であっても，モデルが一貫した判断過程を辿りやすくなるように設計されている．この推論手順の設計は，主に次の二つの観点に基づいている．

一つ目に，要件文は必要な機能や仕様が明示的に記述されていない場合が少なくない．そのため，要件文をそのまま API に対応付けるのではなく，まずサービスの目的を達成するために必要となるコアな機能を明確に導出する必要がある．この段階を設けることで，要件文に含まれる曖昧さを低減し，次のステップの判断基準を明確にすることが可能となる．

二つ目に，抽出されたコア機能を直接 API に対応付けようとすると，候補となる API の数が非常に多くなり，適切な API 選定が困難になる．特に，本研究で対象とするデータセットには多数の API が含まれており，機能が類似した API も多く存在する．この問題を回避するため，本研究では，提供されたデータセットに定義された API カテゴリを中間的なステップとして導入し，コア機能と API カテゴリとの対応付けを行う．これにより，推論の対象範囲を段階的に絞り込み，API 選定を体系的に進めることが可能となる．

以上の考え方に基づき，本研究で定義する推論手順は，以下の 4 段階から構成される．この 4 段階は，要件文の解釈から最終的な API 推薦に至るまでの判断過程を明示的に分離したものであり，各段階がそれぞれ明確な役割を持つ．具体的な推論手順の例を図 11 に示す．

1.　コア機能の抽出

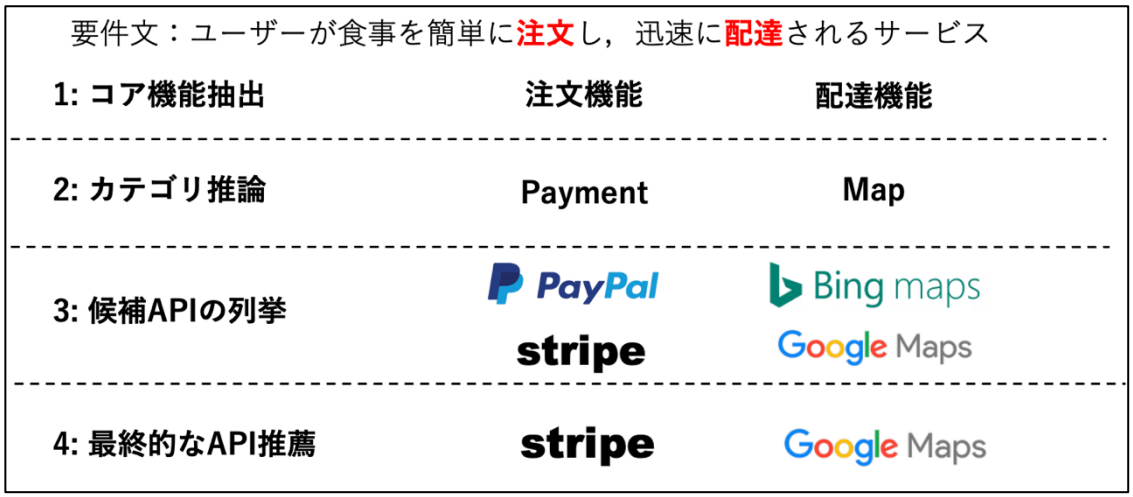

図 11:　トップダウンに基づいた推論手順の例

最初の段階では，要件文に含まれる目的を分析し，サービスを実現するために必要となる中核的な機能を抽出する．この段階では API や実装技術を考慮せず，どのような機能が求められているかという観点から要件文を解釈する．例えば，図 11 の例において，「ユーザが食事を簡単に注文し，迅速に配達されるサービス」という要件文に対しては，「注文」と「配達」という二つの機能がコア機能として抽出される．このようにコア機能を明示的に定義することで，後続の推論ステップにおいて処理対象となる機能単位が明確化される．

2. API カテゴリの推論

抽出された各コア機能に対して，対応する API カテゴリを選定する．コア機能を直接 API に対応付けようとすると，候補となる API の数が非常に多くなり，適切な API 選定が困難になる．そのため，提供されたデータセットに定義された API カテゴリを用い，コア機能と API カテゴリとの対応付けを行う．この段階を導入することで，API 選定における判断の範囲を段階的に絞り込むことが可能となる．図 11 の例において，「注文」というコア機能に対しては「Payment」カテゴリが，「配達」というコア機能に対しては「Map」カテゴリが対応付けられる．

3. カテゴリ毎の候補 API の列挙

API カテゴリが推論された後，各カテゴリに属する API の中から，コア機能に適合すると考えられる複数の候補 API を列挙する．コア機能および API カテゴリが定まっていても，API を即座に決定することは，誤った選定や柔軟性の低下につながる可能性がある．そのため，この段階では，提供されたデータセットに含まれる API の名称，カテゴリ，および説明文を参照し，前段階で抽出されたコア機能に関連性の高い API を列挙する．図 11 の例において，「Payment」カテゴリに対しては「PayPal」と「Stripe」が，「Map」カテゴリに対しては「Google Maps」と「Bing Maps」が候補として列挙される．このように，カテゴリ単位で API 候補を整理することで，後続の最終選定を体系的に行うことができる．

4. 最終的な API の推薦

列挙された候補 API の中から，要件に最も適した API を選定する．この際，抽出されたコア機能を可能な限り網羅する API 構成となるように選定を行うとともに，複数の API を組み合わせて利用する複合サービスであることを前提とし，API 間の役割分担や機能の補完関係を踏まえて推薦を行う．本研究

Follow the reasoning process below:
1. Infer the core functionalities of the mashup service from the given [Requirements].

2. Select relevant categories from [Available Categories] that correspond to the core functionalities inferred in Step1.

3. Select APIs from [Available APIs] within the categories selected in Step 2 whose descriptions align with the core functionalities inferred in Step 1.

4.Propose an optimal combination of APIs identified in Step 3 that covers the core functionalities inferred in Step1 with the minimum number of APIs.

図 12: 推論手順プロンプト

では，各コア機能に対して最適な API をカテゴリごとに選定し，サービス全体として整合性の取れた API 構成を最終的な推薦結果として提示する．図 11 より，「Payment」カテゴリに属する候補 API の中からは「Stripe」を，「Map」カテゴリに属する候補 API の中からは「Google Maps」を選定することで，「注文」および「配達」というコア機能をそれぞれ実現する API 構成が得られる．このように，複数の API を適切に組み合わせることで，要件文に記述されたサービス目的を満たす API 推薦が可能となる．

図 12 に，具体的な推論手順のプロンプトを提示する．ここで示されている Available Categories と Available APIs が提供されたデータセットに含まれる API の情報である．詳細は 5 章で述べる．

4.2 API 推薦における事例生成プロンプト

4.1 節で定義した推論手順に沿った事例を作成するために，事例生成プロンプトを API 推薦タスク向けに具体化する．本研究で用いるデータセットは，要件文，カテゴリ情報，API 名，および API 説明文から構成されるが，推論手順に対応する中間的な推論過程(コア機能抽出，候補 API 列挙)や，各段階における選定理由は含まれていない．そこで，既存データに含まれない中間推論と理由情報を補完することを目的として，図 13 に示す事例生成プロンプトを用いる．本プロンプトは，以下の 7 つの要素から構成される．

1. **指示**

複合サービスの要件文から，API を推薦するタスクの事例を生成する役割を

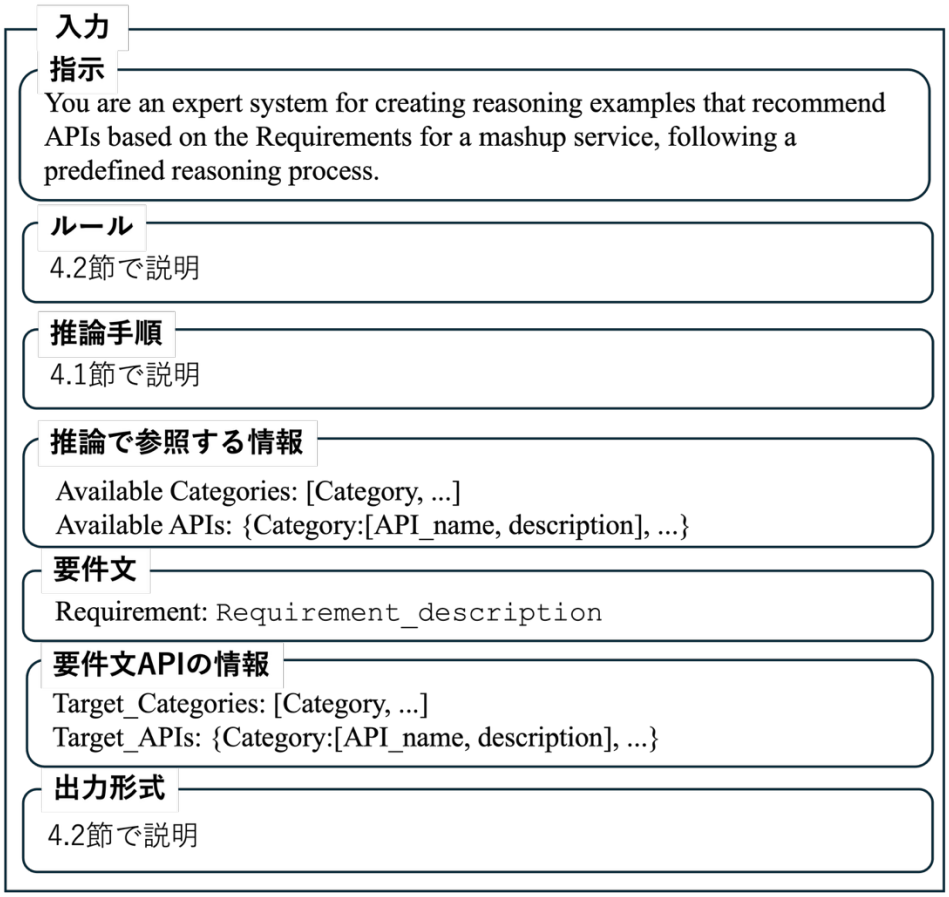

図 13: API 推薦における推論手順に準拠する事例作成プロンプト

定義する．

2. ルール

事例生成時に守るべき制約を明示する要素である．図 14 に，事例作成プロンプトに与えるルールを示す．ここでは，4.1 節で述べた推論手順の各ステップでの制約について定義している．ステップ 2 のコア機能からカテゴリを推論する段階では，要件文に対応するカテゴリのみを出力する．ステップ 3 のカテゴリ推論からカテゴリ毎の API 候補を列挙する段階では，要件文に対応する API を必ず含めた上で，同一カテゴリ内の候補となり得る API を補完推論によって追加する．ステップ 4 の最終的な API 推薦の段階では，要件文に対応する API のみを出力する．これにより，正解として与えられている API 情報を保持したまま，中間的な推論過程と選定理由を補完することが可能となる．

3. 推論手順

###Rules:
- Step 2: output only the [Target_Categories] provided as input.

- Step 3: output all [Target_Apis] provided as input, and you may include a few plausible but incorrect APIs, but they must be selected from the [Available APIs] and belong to the Step2 categories.

- Step4: output only the [Target_Apis].

図 14: ルールのプロンプト

4.1 節で述べた推論手順を組み込む.

4. **推論で参照する情報**
   利用可能な API カテゴリの一覧と，各カテゴリに属する API の名称および説明文を構造化された形式で提示する．モデルはこの情報を参照し，補完推論により中間的な推論過程を生成する．Available_Categories の[Category]には，カテゴリ名のリストを入力し，Available_APIs の辞書は各 API が所属するカテゴリを key として，Value に API 名，その説明文を入れる．
5. **要件文**
   推論手順に準拠した事例を作成したい複合サービスの要件文をここに記述する．Requirement_description に要件文を入力する．
6. **要件文に対応する API の情報**
   5.で入力した要件文に対応する API の情報を与える要素である．カテゴリ情報は Tatget_Category としてリストで与え，API 名とその説明文は，Tatget_APIs として与える．この要素を与えることで，補完推論によって生成される推論過程が，正解 API から逸脱しないよう制御される．
7. **出力形式**
   図 15 に出力形式を示す．出力形式は推論手順に対応した構造を持ち，中間的な推論過程と最終的な API 推薦結果を明示的に分離して出力するよう設計されている．

以上の構成に基づき，本プロンプトの出力と，5.で入力した要件文を組み合わせることで，推論手順に準拠した事例を生成する．図 16 にその一例を示す．図 16 では，Breakout という複合サービスを例としており，当該サービスは Gmail,

Output format:
####reasoning process
1. Core Features:
    ・Feature_name: reason, ...
2. API Categories:
    ・Category_name: reason, ...
3. Candidate APIs per Category:
    ・Category_name:
        ・API_name: reason, ...
    ...
4. Final API Recommendation:
    ・API_name:Reason, ...
###answer
Recommend categories: ['Category_name',...]
Final Recommend APIs:[‘API_name',...]

図 15: 出力形式

Requirement: Breakout is the first modern Workflow Management Software that automates recurring business processes.

###reasoning process###
1. Core Features:
• Automation of Business Processes : reason
• Data Handling and Documentation ; reason
:
2. API Categories:
• Email: reason, Spreadsheets: reason, Storage: reason, Tools: reason
3. Candidate APIs per Category:
Email:
mailchimp: reason, mandrill: reason, sendGrid: reason, gmail: reason
:
4. Final API Recommendation:
Gmail : reason, google-sheets: reason, google-drive: reason, slack-web: reason

###answer###
Recommend categories: [‘Email’, ’Spreadsheets’, ’Storage’, ’Tools’]
Final recommend APIs: [‘Gmail’, ’google-sheets’, ’google-drive’, ’slack-web’]

図 16: 事例の具体例

Google Sheets，Google Drive，Slack Web の 4 つの API を組み合わせて構成されている．これらの API はそれぞれ，email，spreadsheets，storage，tools の各カテゴリに属する．これらの情報を事例作成プロンプトに与えることで，カテゴリ選定および最終 API 提示の段階においては，各 API の選定理由が補完される．さらに，コア機能推論の段階では，与えたカテゴリ情報と要件文を対応付ける形で，コア機能名およびその推論理由が補完される．加えて，API 列挙段階では，最終提示 API およびカテゴリ情報をもとに，その他の類似した API が列挙され，それぞれに選定理由が補完される．

4.3 API 推薦におけるガイド付推論連鎖型プロンプト

本節では，3 章で提案したガイド付推論連鎖型プロンプトの枠組みに基づき，API 推薦タスクにおいて推論時に用いるプロンプト構造を示す．図 17 に，API 推薦において実際に用いるプロンプトの全体構造を示す．本プロンプトは，4.2 節で述べた事例作成プロンプトと同一の枠組みを共有する一方で，推論時には正解 API に関する情報を入力として与えない構成となっている．図 17 に示すプロンプトは，以下の 6 要素から構成される．なお，各要素の一般的な役割は 3 章

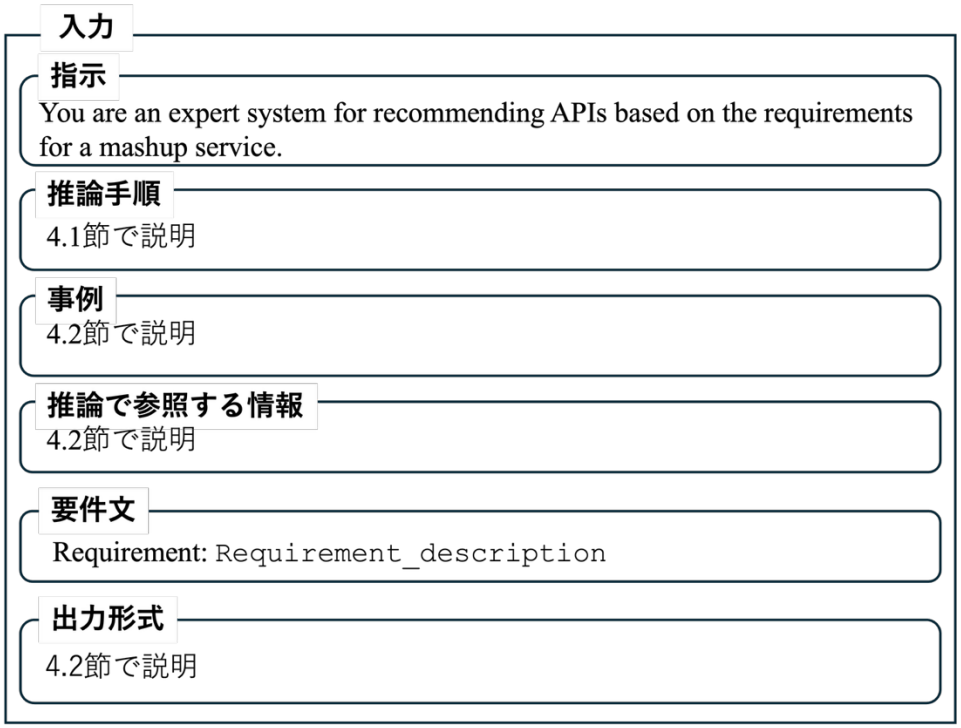

図 17: API 推薦プロンプト

で述べた設計思想に従うため，本節では API 推薦タスクにおいて具体的に与える内容に焦点を当てて説明する．

1. **指示**
   複合サービスの要件文に基づき，複数の API を選定して推薦するように指示する．
2. **推論手順**
   4.1 節で定義した API 推薦における推論手順を記述する．
3. **事例**
   4.2 節で生成した，推論手順に準拠する事例を Few-shot として埋め込む．
4. **推論に必要な情報**
   4.2 節で述べた推論対象となる API の知識源として，利用可能な API カテゴリの一覧を Available Category として，各カテゴリに属する API 名および API 説明文を Available APIs として提示する．
5. **要件文**
   ユーザが記述した複合サービスの要件文を入力として与える．
6. **出力形式**
   4.2 節で定義した出力形式を用いる．

第5章 評価

本章では，第 4 章で構築した API 推薦におけるガイド付推論連鎖型プロンプトを用いて評価を行う． 5.1 節では，実験に用いるデータセットについて説明するとともに，実行環境，比較手法，および評価方法を述べる．5.2 節では，実験結果を示し，5.3 節では，アブレーションテストを通して提案手法の有効性を検証する．

5.1 評価手法

5.1.1 実験データ

本研究では，programmableWeb[1] に記載されている複合サービスデータ 6467 件，API のデータ 22469 件を用いる．図 18 に，複合サービスの例を示す．複合サービスデータは，複合サービス名をキーとして，人手で付与されたメインカテゴリ，サブカテゴリ，複合サービスの説明文，複合サービスに使用されている API の名前で構成されており，Json 形式で記述されている．取得した複合サービス

```
"artist-cloud-igoogle-gadget": [
   [
    "music",
    "video",
    "widgets",
    "search"
   ],
   "iGoogle gadget of the popular Music Artist Cloud mashup for music
    discovery based on similar artists.",
   [
    "youtube",
    "lastfm",
    "google-gadgets",
    "google-app-engine"
   ]
]
```

図 18: 収集した複合サービス例

[1]https://www.programmableweb.com

```
"複合サービス名": [
  [
   "メインカテゴリ",
   "サブカテゴリ"
  ],
  "複合サービスの説明文",
  [
   "使用しているAPI"
  ]
]
```

図 19: 複合サービスのデータ形式

```
"youtube": [
  [
   "video",
   "media"
  ],
  "The Data API allows users to integrate their program with YouTube and
   allow it to perform many of the operations available on the website. ...",
  [
   "google"
  ],
  [
   "connor-s",
   "marta-mateos-gonzalez",
   "niki-lai"
  ]
]
```

図 20: 収集した API の例

のデータ構造のフォーマットを図 19 に示す．次に，図 20 に API データの例を示す．API データは，API 名をキーとし，ProgrammableWeb において人手で付与されたメインカテゴリ，サブカテゴリ，API の説明文，API の提供者名，API の利用者名で構成されている. API のデータ構造のフォーマットを図 21 に示す. 収集したデータには一部の欠損が含まれていたため，適切なデータのみを抽出

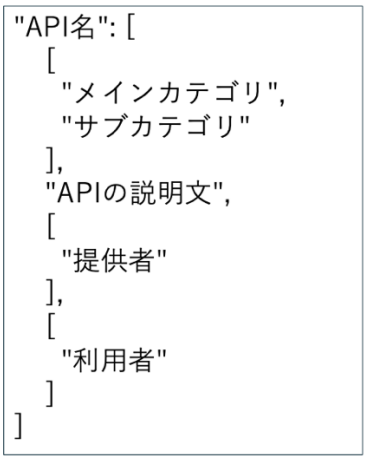

図 21: 収集した API のデータ形式

するためにフィルタリングを行った．具体的には，複合サービスの選別をしてから API データの選別の二段階に分けてフィルタリングを行った．まず，複合サービスデータについては，2 つ以上の API を構成している複合サービスのみを対象とした．さらに，複合サービスに利用されている API のデータが欠損していない場合に限り分析対象とした．次に，API データについてはフィルタリング後の複合サービスに使用されている API のみを対象とした．結果，複合サービス 1,497 件，API645 件，115 カテゴリで評価を行った．

5.1.2 比較手法

提案手法の有効性を検証するため， 5 つの代表的なベースライン手法を設定し，API 推薦性能の比較を行った．いずれの手法においても，評価条件を統一するため，4 章で述べた API 推薦タスクと同一の指示文および推論に必要な情報をプロンプトに付与した．具体的なプロンプト構成を図 22 に示す．出力形式については，各プロンプト手法において推論過程の表現形式を統一することが出来ないため，推論プロセス部分は自由形式とした．一方で，評価する推薦結果については，明示的に出力する箇所を全手法で共通化した．各ベースライン手法の概要を以下に示す．各手法の基本的な考え方については 3 章で説明しているため，本節では API 推薦タスクへの適用における設定上の差分のみを述べる．

- **Zero-Shot**
  モデルの入力として指示と要件文のみを使用する．事例や推論過程を一切含めない．
- **Zero-Shot-CoT**

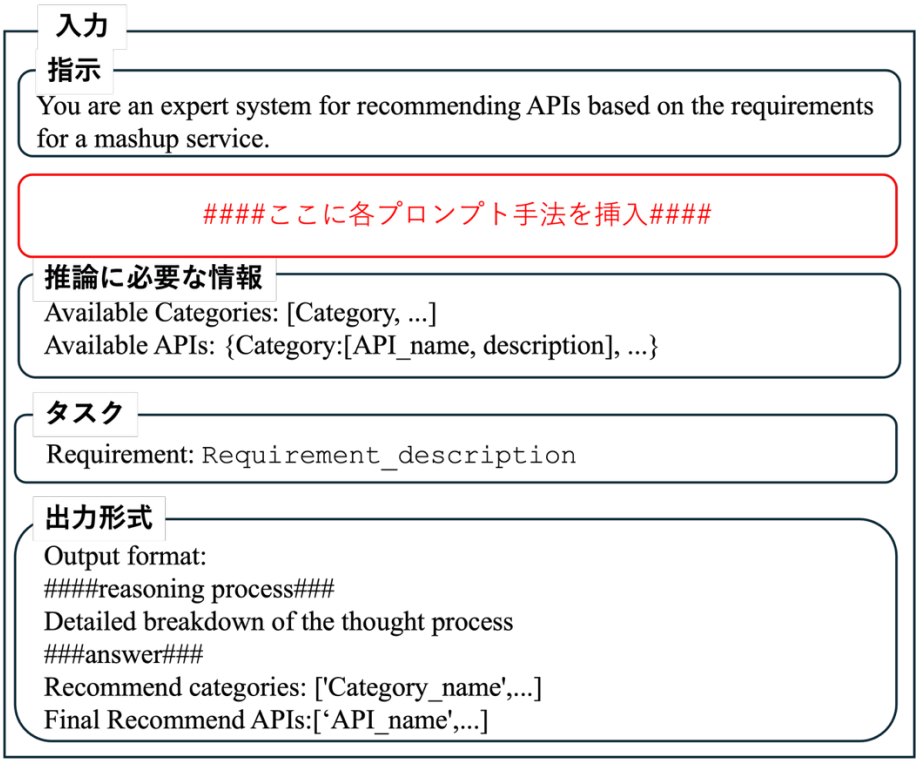

図 22: 比較手法フォーマット

Zero-Shot と同様の指示と要件文を与える．

- **Few-Shot**
  少数の事例をプロンプトに含めるが，事例には要件文とそれに対応する最終的な API およびカテゴリのみを含め，推論過程は含めない．
- **Few-Shot-CoT**
  Few-Shot プロンプティングにおいて，各事例に推論過程を含めることで，モデルに推論の進め方を例示する手法である．事例の一例を図 23 に示す．同図では，4 章の図 16 で説明した推論手順に準拠した事例と同一の要件文を使用している．Few-Shot-CoT では，各推論段階における判断基準や制約条件は明示されておらず，推論手順は事例から暗黙的に判断する必要がある．
- **Plan and Solve**
  Zero-Shot と同様の指示と要件文を与える．

Requirement: Breakout is the first modern Workflow Management Software that automates recurring business processes.

###reasoning process###
The service aims to automate recurring business processes, which implies managing workflows, handling structured data, and storing related documents.
Automation often requires notifying users or systems, so communication mechanisms are necessary.
To support these functions, data organization, file storage, and integration or notification tools are also required.

Based on these considerations, APIs related to email communication, spreadsheets, storage, and general tools are suitable candidates.
Among commonly used services, selecting flexible and widely adopted APIs is appropriate for building such a workflow management system.

###answer###
Recommend categories: ['Email', 'Spreadsheets', 'Storage', 'Tools']
Final recommend APIs: ['Gmail', 'google-sheets', 'google-drive', 'slack-web']

図 23: Few-shot で使用する事例の一例

5.1.3 利用したモデル

提案手法のプロンプト構成が特定の LLM に依存せず機能すること検証するため 4 種類の LLM を用いて比較実験を行った. 具体的には, GPT-4.1-mini, GPT-oss-120B, DeepSeek-R1:70B, Llama4:70B を用いて評価した. GPT-5-mini は API 経由で利用した. 一方, GPT-oss-120B[2], Deepseek-R1:70B[3], Llama-4:70B[4] については, ollama 上で提供されているモデルを使用した. 以下にモデルの詳細を述べる.

・GPT-4.1-mini

OpenAI により提供されている GPT-4 系列の軽量版である. パラメータ数は公開されていない. 本モデルは最大 1,047,576 トークンのコンテキストウィンドウをサポートし, 1 回の推論において最大 32,728 トークンの出力が可能である. また, 2024 年 5 月 1 日までのデータで学習している.

---

[2] https://huggingface.co/openai/gpt-oss-120b

[3] https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Llama-70B

[4] https://huggingface.co/meta-llama/Meta-Llama-3-70B

A conversation between User and Assistant. The user asks a question, and the Assistant solves it. The assistant first thinks about the reasoning process in the mind and then provides the user with the answer. The reasoning process and answer are enclosed within <think> </think> and <answer> </answer> tags, respectively, i.e., <think> reasoning process here </think> <answer> answer here </answer>. User: prompt. Assistant:

図 24: 学習テンプレート

・GPT-oss-120B[14]

OpenAI により公開された LLM である．Mixture-of-Experts(以下，MoE) Transformer を採用した自己回帰型モデルであり，総パラメータ数は約 116.8B である．一方，推論時に各トークンあたりで有効化されているパラメータ数は約 5.1B に抑えられている．事前学習は STEM，コード，一般知識を中心としたデータを用いて行われている．本モデルは最大 131,072 トークンのコンテキストウィンドウをサポートし，1 回の推論において最大 131,072 トークンの出力が可能である．また，2024 年 6 月 1 日までのデータで学習している．

・Deepseek-R1:70B[15]

DeepSeek-AI により公開されたオープンソースの LLM である．本モデルは DeepSeek-R1 が生成した推論データを用いて，70B 規模のモデルに対して教師あり学習することで構築している．DeepSeek-R1 の構築過程では，初期段階として Deepseek-R1-Zero が用いられる．DeepSeek-R1-Zero は図 24 に示すテンプレートを用いて学習される．このテンプレートでは，”prompt”部分に数学やプログラミングなど正解が明確な問題を入力する．モデルの出力は，思考過程を表す”think”と最終的な解答を表す”answer”から構成される．学習においては，生成された”answer”のみを評価対象とし，その評価に基づいてモデルの出力を更新する．この学習を繰り返すことで，”think”に含まれる長い思考過程を伴う推論能力がモデルに獲得される．その後，この出力結果と，少量の高品質な思考過程データで教師あり微調整を行うことで DeepSeek-R1:70B が得られる．

・Llama4:70B[16]

Meta により公開された LLM である MoE Transformer を採用した自己回帰型モデルを用いている．事前学習には，大規模なテキストデータに加え，画像データを用いており，多言語テキストの理解と生成および画像からの推論に対

応している．本モデルは長文コンテキストを扱う能力を有するが，詳細なパラメータ数や学習データの最終時点については公開されている情報では明示されていない．

5.2 結果

提案手法により生成される出力を評価する指標として Precision 値，Recall，F1 を用いる．各手法で定義した出力形式のカテゴリ推薦と API 推薦を評価対象とする．正解データは，ProgrammableWeb から取得した複合サービスデータに対応する API と，それらの API に付与されているカテゴリ情報を基に作成した．なお，カテゴリについては，5.1 節で説明した API データ構造のうち，メインカテゴリのみを使用した．事例を用いる手法において，事例数は 6 件に統一した．

表 1: カテゴリ推論の検証結果

		Precision	Recall	F1
GPT-4.1	Zero-Shot	0.631	**0.693**	0.661
	Zero-Shot CoT	0.617	0.673	0.644
	Few-Shot	0.660	0.680	0.669
	Few-Shot CoT	0.617	**0.693**	0.653
	Plan and Solve	0.584	0.661	0.620
	Ours	**0.681**	0.685	**0.683**
GPT-OSS	Zero-Shot	0.291	0.345	0.316
	Zero-Shot CoT	0.307	0.364	0.333
	Few-Shot	0.290	0.346	0.315
	Few-Shot CoT	0.259	0.315	0.284
	Plan and Solve	0.293	0.348	0.318
	Ours	**0.332**	**0.383**	**0.356**
LLaMA	Zero-Shot	0.255	0.408	0.313
	Zero-Shot CoT	0.258	0.418	0.319
	Few-Shot	0.245	0.389	0.301
	Few-Shot CoT	0.135	0.196	0.160
	Plan and Solve	0.268	0.404	0.322
	Ours	**0.278**	**0.433**	**0.339**
DeepSeek	Zero-Shot	0.287	0.377	0.326
	Zero-Shot CoT	0.306	**0.382**	0.340
	Few-Shot	0.174	0.218	0.194
	Few-Shot CoT	0.213	0.263	0.235
	Plan and Solve	0.281	0.359	0.315
	Ours	**0.316**	0.372	**0.342**

表 2: API 推論の検証結果

		Precision	Recall	F1
GPT-4.1	Zero-Shot	0.345	0.413	0.376
	Zero-Shot CoT	0.340	0.398	0.367
	Few-Shot	0.433	**0.456**	0.444
	Few-Shot CoT	0.413	0.427	0.420
	Plan and Solve	0.343	0.381	0.361
	Ours	**0.460**	0.441	**0.450**
GPT-OSS	Zero-Shot	0.047	0.029	0.036
	Zero-Shot CoT	0.051	0.031	0.038
	Few-Shot	0.046	0.028	0.035
	Few-Shot CoT	0.046	0.029	0.036
	Plan and Solve	0.051	0.034	0.041
	Ours	**0.070**	**0.050**	**0.058**
LLaMA	Zero-Shot	0.074	0.091	0.082
	Zero-Shot CoT	0.058	0.076	0.066
	Few-Shot	0.070	0.054	0.065
	Few-Shot CoT	0.045	0.076	0.057
	Plan and Solve	0.085	0.084	0.084
	Ours	**0.086**	**0.105**	**0.095**
DeepSeek	Zero-Shot	0.060	0.070	0.065
	Zero-Shot CoT	0.062	0.062	0.062
	Few-Shot	0.066	0.062	0.064
	Few-Shot CoT	0.062	0.053	0.057
	Plan and Solve	0.066	0.077	0.071
	Ours	**0.080**	**0.088**	**0.084**

この時，各要件文でランダムに事例を選定するが，異なる手法間で選択される事例は統一している．カテゴリ推論の結果を表 1 に示す．GPT および DeepSeek では，Recall の値が一部のベースライン手法と比較して低下する傾向が見られたものの，提案手法(Ours)は全てのモデルにおいて，他のベースライン手法を上回る精度を記録した．特に，GPT-4.1-mini を用いた場合，提案手法は F1 が 0.683 と最も高い値を達成している．また，LLaMA モデルでは，Precision，Recall，F1 の全ての指標において，提案手法がベースライン手法を上回る結果となった．API 推論の結果を表 2 に示す．GPT では，カテゴリ推論と同様に，Recall の値が一部のベースライン手法と比較して低下する傾向が見られた．一方で，LLaMA および DeepSeek では，すべての指標において提案手法がベースライン手法を上回る結果となった．

以上のカテゴリ推論および API 推論の結果から，構造化された推論プロセスは自由形式の推論プロセスよりも効果的であり，その設計がモデル間で一貫して有効に機能することが示された．

5.3 アブレーションテスト

提案手法のプロンプト設計の有効性を検証するため，各構成要素が推論性能に与える影響を分析した．本分析では，提案手法のプロンプトを基準とし，特定の構成要素を除去した 2 種類のアブレーションプロンプトを設計した．図 25 に各プロンプトの構成を示す．Few-Shot Ablation では，提案手法のプロンプトから推論手順を除去することで，構造化された推論手順の有無が推論性能に与える影響を評価する．一方，Zero-Shot Infer では，事例を除去することで，Few-Shot 設定の有無が推論性能に与える影響を評価する．

カテゴリ推薦におけるアブレーション結果を表 3 に示す．GPT では，Few-Shot Ablation において Recall が高い値を示す一方で，Precision が低下する傾向が見られた．これに対し，提案手法は F1 において最も高い値を達成している．また，LLaMA および DeepSeek においても，提案手法は両アブレーション設定を上回る F1 を記録しており，構造化された推論手順および事例提示を組み合わせたプロンプト設計の有効性が確認できる．

API 推薦におけるアブレーション結果を表 4 に示す．全ての手法において，提案手法は両アブレーション設定を上回る F1 を記録しており，構造化された推

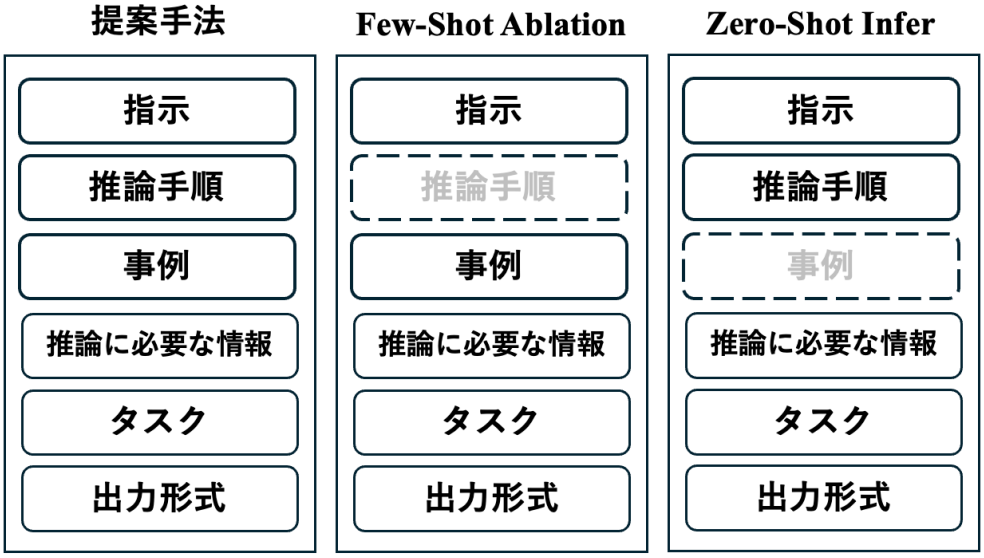

図 25: アブレーションプロンプトの構成

表 3: カテゴリ推薦におけるアブレーション結果

		Precision	Recall	F1
GPT-4.1	Few-Shot ablation	0.677	**0.688**	0.682
	Zero-Shot Infer	0.538	0.735	0.621
	Ours	**0.681**	0.685	**0.683**
GPT-OSS	Few-Shot ablation	0.258	0.314	0.283
	Zero-Shot Infer	0.258	0.315	0.284
	Ours	**0.332**	**0.383**	**0.356**
LLaMA	Few-Shot ablation	0.271	0.404	0.324
	Zero-Shot Infer	**0.293**	0.392	0.335
	Ours	0.278	**0.433**	**0.339**
DeepSeek	Few-Shot ablation	0.302	0.363	0.330
	Zero-Shot Infer	**0.320**	0.365	0.341
	Ours	0.316	**0.372**	**0.342**

表 4: API 推薦におけるアブレーションテスト結果

		Precision	Recall	F1
GPT-4.1	Few-Shot ablation	0.418	0.424	0.421
	Zero-Shot Infer	0.340	0.396	0.366
	Ours	**0.460**	**0.441**	**0.450**
GPT-OSS	Few-Shot ablation	0.047	0.028	0.035
	Zero-Shot Infer	0.046	0.028	0.035
	Ours	**0.070**	**0.050**	**0.058**
LLaMA	Few-Shot ablation	0.055	0.073	0.063
	Zero-Shot Infer	0.063	0.044	0.052
	Ours	**0.086**	**0.105**	**0.095**
DeepSeek	Few-Shot ablation	0.073	0.091	0.081
	Zero-Shot Infer	0.069	0.072	0.070
	Ours	**0.080**	**0.088**	**0.084**

論手順および事例提示を組み合わせたプロンプト設計の有効性が確認できる．以上のアブレーション結果から，提案手法においては，構造化された推論手順および推論手順に準拠した事例の双方が推論性能の向上に寄与しており，これらの構成要素を組み合わせたプロンプト設計が有効であることが確認された．

第6章 考察

6.1 ハルシネーション分析

LLM を用いた推論タスクでは，入力として与えられていない情報や事実に基づかない内容を出力するハルシネーションが生じる場合がある．本研究では，Available Category および Available API に含まれない要素が出力された場合をハルシネーションと定義し，その発生状況を分析した．本分析では，すべての手法において GPT-4.1-mini を用い，提案手法と同一の 1497 件の複合サービスを対象として比較を行った．使用した事例は Random である．ハルシネーションのカウント方法としては, 要件文ごとに, 出力結果に 1 つでも Available Category または Available API に含まれない要素が含まれていた場合，当該要件文を 1 件としてカウントした.

6.1.1 手法間比較による発生頻度

表 5 に，各手法におけるカテゴリおよび API のハルシネーション発生要件文数，ならびに両者が同一要件文内で同時に発生した件数を示す. いずれの手法においてもカテゴリに関するハルシネーションは少数にとどまっている一方，API に関するハルシネーションは手法間で大きな差が見られる．Few-Shot CoT では API に関するハルシネーションが比較的少なく，事例提示によって API 選定時の誤りが一定程度抑制されたと考えられる．これに対し，Zero-Shot および Plan and Solve では 140 件以上と多く発生していることから，事例を与えずに自由に推論を生成させる手法では，多数の API から選定する段階においてハルシネーションが生じやすいことが示唆される．これらの API ハルシネーションの多くはカテゴリの誤りを伴わずに単独で発生しており，カテゴリと API のハルシネーションが同時に発生した要件文は，Zero-Shot では確認されず，他の手法においても最大で 2 件にとどまった. 本研究で提案する手法においても，同時発生は 1 件のみであり，カテゴリのみのハルシネーションは 0 件，API のみのハルシネーションは 54 件と，すべての手法の中で最も少ない発生件数を示している．以上より，ハルシネーションは推論全体が一度に崩壊する形で生じるのではなく，主として最終的な API 選定段階に局所的に発生しやすく，提案手法はこの段階における誤りを最も効果的に抑制できていることが分かる.

表 5: カテゴリと API ハルシネーションの発生件数

	Category のみ	API のみ	同時
Zero-Shot	4	145	0
Zero-Shot CoT	5	76	1
Few-Shot CoT	7	57	2
Plan and solve	3	139	2
Ours	0	54	1

表 6: ハルシネーション発生要件文に限定した平均個数

	カテゴリ総数	API 総数	カテゴリ平均	API 平均
Zero-Shot	4	235	1.00	1.62
Zero-Shot CoT	6	98	1.00	1.27
Few-Shot CoT	10	65	1.11	1.10
Plan and solve	6	195	1.20	1.38
Ours	1	58	1.00	1.06

6.1.2 1 要件文内での発生規模

ハルシネーションが発生した要件文に限定し, 1 要件文あたりに発生したカテゴリおよび API のハルシネーション個数の平均を算出した．表 6 よりカテゴリに関する平均発生数はいずれの手法でも概ね 1 件であり, 同一要件文内で多発する傾向は見られなかった．一方，API に関しては事例を与えない手法では，1 要件文内で複数のハルシネーションが生じやすい傾向が確認された．これに対し，提案手法では平均 1.06 件と最小であり，ハルシネーションが発生した場合でも影響が限定的であることが分かる．また, 付録に各要件文のハルシネーション件数を載せている．

6.2 事例選定による精度変化

本節では，提案手法において，プロンプトに与える事例の選定方法が推論精度に与える影響を分析する．事例選定にあたっては, 要件文同士の意味的類似度に基づく単純な方針を採用した．具体的には，要件文を BERT モデルによりベクトル化し，入力要件文と事例で使用した要件文とのコサイン類似度に基づいて順位付けを行った．すべての戦略において，GPT-4.1-mini を用い，提案手法と同一の 1497 件の複合サービスを対象として, カテゴリ推薦および API 推薦における精度変化を調査した．

6.2.1 事例選定戦略と精度への影響

本研究では，事例選定戦略の違いが推論精度に与える影響を明らかにするため，以下の 4 つの戦略を比較した．

・Random Strategy

事例をランダムに 6 件選択する．本研究におけるベースライン．

・Nearest Strategy

入力要件文と意味的に最も類似度の高い 6 件の事例を選択する．

・Furthest Strategy

入力要件文と意味的に最も類似度の低い 6 件の事例を選択する．

・Nearest-Farthest Strategy

入力要件文と最も類似した上位 3 位，最も低い下位 3 件の事例を組み合わせて選択する．

カテゴリ推薦における結果を表 7 に示す．F1 に着目すると，Nearest Strategy が 0.688 と最も高く，次いで Random Strategy，Nearest-Farthest Strategy の順となった．一方，Furthest Strategy は Precision，Recall，F1 のすべての指標において最も低い値を示している．この結果から，入力要件文と意味的に乖離した事例を提示した場合，カテゴリ推論の精度が大きく低下することが分かる．

次に，API 推薦における結果を表 8 に示す．API 推薦においても，Nearest Strategy がすべての指標で最も高い値を示した．Random Strategy と比較する

表 7: カテゴリ推薦における選択事例による精度変化

	Precision	Recall	F1
Random	**0.681**	0.685	0.683
Nearest	0.643	**0.741**	**0.688**
Furthest	0.592	0.694	0.639
Nearest-Farthest	0.633	0.730	0.678

表 8: API 推薦における選択事例による精度変化

	Precision	Recall	F1
Random	0.460	0.441	0.450
Nearest	**0.538**	**0.577**	**0.557**
Furthest	0.410	0.424	0.417
Nearest-Farthest	0.493	0.531	0.511

と，F1 は約 0.11 向上しており，事例選定が API 推薦精度に与える影響が大きいことが確認できる．一方，Furthest Strategy はカテゴリ推薦と同様に低い精度を示しており，意味的に不適切な事例が提示された場合，最終的な API 選定においても誤りが生じやすいことが分かる．

以上の結果から，提案手法では，推論手順そのものの設計に加えて，入力要件文と意味的に整合した事例が提示されるかどうかが，推論精度に大きく影響することが明らかとなった．特に，意味的に類似した要件文に基づく事例を与えることで，モデルはコア機能の解釈，カテゴリ選定，API 選択といった各推論段階を，事例に沿って適切に適用できていると考えられる．

6.2.2 実例に基づく分析

ショット選択による推薦結果の影響を実際の複合サービスを対象として分析を行う．本節では，「Holidayen」という旅行計画を支援する複合サービスを分析対象とする．Holidayen は，「300 以上の目的地に対応した旅行プラン生成サービスを提供し，旅行計画に要する時間を数週間から数分に短縮する」という要件文を持つ複合サービスである．本サービスは，多機能な API 推薦が求められる点から，事例選定戦略の影響を分析する対象として適している．付与されているカテゴリは Mapping，Photo，Travel の 3 つであり，使用されている API は Google Maps，Flickr，Viator である．表 9 に，Nearest Strategy および Furthest Strategy において，プロンプトに選択された事例に含まれているカテゴリおよび API の内訳を示す．表中の「～件」という表記は，当該カテゴリまたは API が何件の事例に含まれていたかを示しており，件数の記載がないものについては，1 件の事例に含まれていたことを意味する．

Nearest Strategy では，mapping，photo，travel など，旅行計画や観光体験と関連性の高いカテゴリが多く含まれていることが分かる．API についても，google-maps，flickr，booking.com，skyscanner など，旅行に関連性の高い API が中心となっている．その結果，カテゴリ推論として travel，mapping，recommendations, photo が出力され，API 推薦においても google-maps, flickr, viator，yelp-fusion といった正解 API を含む結果が得られた．特に，viator は選択された事例内に含まれていない API であるにもかかわらず推薦されている．このことから，モデルが事例を単純に表層的に模倣しているのではなく，要件文の内容と，事例から得られた推論構造を組み合わせて判断していることが示唆される．

Furthest Strategy では，入力要件文と意味的に最も類似度の低い事例が選択されるため，選択された事例に含まれるカテゴリおよび API は，旅行計画とは直接的な関連性の低いものが多く含まれている．カテゴリには social, telephony, education, reference などが含まれており，API についても，一部に旅行関連 API は含まれるものの，直接的な関連性の低いものが多い．その結果，カテゴリ推薦として travel，booking，location，recommendations，events，ecommerce などが出力された一方で，正解カテゴリである photo は推論されなかった．また，API 推薦においても，bookingcom-content，eventbrite，google-maps-places，skyscanner-live-pricing，tripit などが選択され，正解 API である Flickr および Viator は含まれなかった．

これらの結果から，事例選定においては，単に事例を提示するだけでなく，事例内に含まれる情報の性質が推論結果に直接的な影響を与えることが分かる．すなわち，入力要件文と意味的に整合した事例が提示された場合には，モデルは推論手順を適切に適用できる一方で，意味的に乖離した事例が提示された場合には，推論が不適切な方向に誘導される可能性が高くなることが示された．

6.3 推論に失敗した複合サービスの分析

本節では，提案手法において最終的に正しい API を提示できなかった複合サービスを対象とし，推論手順のどの段階で失敗が生じているのかを分析する．

6.3.1 推論段階別に見た失敗ケース

本研究では，第 4 章で述べた推論手順に基づき，最終的に正しい API を提示

表 9: 選ばれた事例の詳細

	カテゴリ	API
Nearest	mapping :4 件, social :3 件, photo: 3 件, travel, booking, url-shortener,video, mapping, advertising	google-maps: 5 件, flickr: 3 件, facebook: 2 件, bookingcom-content, skyscanner-live-pricing, foursquare, youtube, google-adsence
Furthest	social: 2 件, mapping: 2 件, photos: 2 件 telephony, messaging, education, reference, video, ecommerce, music, content, movies	twitter: 2 件, google-maps: 2 件 件,flicker:2 件, new-york-times-movie-reviews, box, friendfeed, amazon-product-advertising, youtube, lastfm, musicvrainz, geonames, donorschoose, facebook, twillo, twillo-sms

できなかった推論結果について，推論過程のどの段階で誤りが生じたかを Gpt4.1-mini の結果を用いて分析する．評価した 1497 件中，最終的に正しい API を提示できた推論（F1= 1.0）は 219 件であり，それ以外の 1278 件を分析対象とした．これらの失敗ケースについて，推論手順に基づき以下のように失敗要因を定義した. 正解カテゴリとは要件文に利用されている API のカテゴリ名である．

1. **コア機能の抽出**
   推論されたコア機能が要件文の意図と一致せず，正解カテゴリに対応づけられない場合
2. **カテゴリ推論**
   推論されたコア機能が正解カテゴリに対応づけられるにもかかわらず，正解カテゴリが選択されていない場合
3. **候補 API の列挙**
   正しいカテゴリが選択されている一方で，API 列挙段階で正解 API が出現していない場合
4. **最終的な API 推薦**
   最終的に提示された API が誤っている一方で，API 列挙段階では正解 API が含まれている場合

なお，コア機能抽出の失敗とカテゴリ選択の失敗は結果のみから機械的に判定することが困難である．そのため，本研究では LLM を用いた補助的な判定を行った．具体的には，要件文，モデルの推論出力，および正解カテゴリを入力とし，推論されたコア機能が正解カテゴリに自然にマッピング可能かどうかを gpt-4.1-mini で判定させた. 正解カテゴリに対応可能なコア機能が推論されている場合はカテゴリ選択の失敗，対応できない場合はコア機能抽出の失敗として分類した．以上の手法により分類した結果を以下に示す．

・コア機能抽出の失敗: 462 件
・カテゴリ選択の失敗: 55 件
・API 列挙段階の失敗: 530 件
・最終 API 提示段階の失敗: 231 件

この結果から，特にコア機能抽出の失敗と API 列挙段階の失敗が多数を占めていることがわかる．コア機能抽出段階での失敗が多かった理由として，要件文の曖昧性によりユーザ意図を一意に抽象化することが難しい点があるのではないかと考えられる．API 列挙段階の失敗の多かった理由として，推論手順の定義

の曖昧性があったのではないかと考える. API の説明文から判断するという指示のみのため，列挙段階の基準をより詳細に定義する必要があると考える.

6.3.2 全手法が推論に失敗した複合サービス

提案手法を含む全ての手法が推論に失敗した複合サービスについて考察する. 対象とする複合サービスは「backup-box」という複数のクラウドストレージ間でデータを安全かつ自動的に移動してバックアップできるサービスである. backup-box は，「We move your stuff from anything to anywhere」という要件文を持つ複合サービスである．付与されているカテゴリは storage, content の 2 つであり，使用されている API は Google Drive, Amazon-s3, Dropbox, Box, SugarSync である．しかし，全てのモデルはこの要件文を正しく解釈できなかった．具体的には，文中の，move という表現をデータの移動ではなく，物理的な輸送と誤解し，その結果，gpt-4.1-mini を用いた提案手法では，カテゴリ推論として shipping や transportation を選択した．さらに，推薦された API も fedex, ups-shipping, DHL, uship, yahoo-traffic 等の物流や交通機関の API に偏り，本来選択されるべきストレージ系の API は一切含まれなかった．この失敗の要因として，要件記述の曖昧さが挙げられる．特に，move your stuff という表現は抽象的であり，一般的にはデータの移動ではなく，荷物を物理的に移動される行為として解釈されやすい．さらに, anything to anywhere という表現が加わることで，要件全体が引越しや物流を含む複合的な輸送サービスであると考えられてしまう．この解釈の誤りは，その後の推論過程全体に影響を及ぼし，誤ったカテゴリ選択と API 推薦が連鎖的に発生したと考えられる．これらの問題を緩和するためには，推論を開始する前段階において，ユーザに対して簡潔な確認質問を行い，要件文に含まれる曖昧性を解消するステップを導入することが有効であると考えられる．さらに，単一の解釈に基づく提案に限定するのではなく，複数の可能な解釈を考慮し，それぞれに対応した異なる推薦結果を提示するよう拡張することで，誤解釈に起因する失敗を低減できる可能性がある.

6.4 要件文が推薦精度に与える影響

本節では，要件文の性質と API 推薦精度との関係について考察する．要件文の性質を表す指標として要件文長を用い，要件文長と推薦精度との関係を定量的に評価するため，ピアソンの相関係数および対応する p 値を算出した．具体的には，要件文長とカテゴリレベルの F1 スコア，ならびに要件文長と最終提示

API の F1 スコアとの間の相関をそれぞれ求め，両者の関係性を検証した．その結果，要件文長とカテゴリレベル F1 スコアの間には統計的に有意な負の相関が確認された（Pearson の相関係数 r= −0.075, p= 0.003）．しかしながら，相関係数の絶対値は極めて小さく，要件文長がカテゴリ分類性能に与える影響は限定的であることが示唆される．これは，要件文が長くなることで情報量が増加する一方，冗長な記述や複数の要求が混在することにより，カテゴリ判定がわずかに不安定になる可能性を反映していると考えられる．また，要件文長と API レベルの最終 F1 スコアとの間には有意な相関は認められなかった（r= 0.028, p= 0.28）．この結果は，要件文の長さそのものは，最終的な API 推薦精度にほとんど影響を与えないことを示している．すなわち，本研究で用いた推薦モデルは，要件文が短文であっても長文であっても，要件の本質的な意味を抽象化し，適切な API を推薦できていると考えられる．以上の結果から，要件文長はカテゴリレベルの分類性能に対してはごく弱い影響を持つものの, API 推薦という最終タスクにおいては支配的な要因ではないことが明らかとなった．このことは，API 推薦精度の向上には，要件文の長さよりも，要件の内容が重要であることを示唆している．

次に，精度の上位 10 件および下位 10 件を比較した結果，要件文が技術に基づいた記述されているか，サービスの紹介文として記述されているかが，推薦精度に大きく影響していることが示唆された．精度が高かった要件文では，実現すべき機能が操作レベルで具体的に記述されていた．また，複数の機能や処理の関係が文章中に含まれている点も特徴的であった．例えば，ある機能の結果が次の機能を引き起こすといった因果関係や，複数の機能を組み合わせてサービスを構成していることが明示されており，複合サービスとしての構造が自然言語から読み取れる．一方，精度が低かった要件文では，サービス全体の概要や価値を説明する表現が中心となっており，利用体験や結果に焦点が当てられている場合が多く見られた．これらの要件文では，抽象的な表現が多用され，具体的な操作や機能の範囲が明確でないため，コア機能の抽出段階で解釈の揺らぎが生じやすかった．また，機能間の関係や役割分担が記述されていないため，API カテゴリ推論および最終的な API 推薦に必要な設計情報が不足していたと考えられる．以上の結果から，複数の機能やその関係が読み取れる要件文することが重要であると考えられる．

第7章 まとめ

本研究では，自然言語で記述された複合サービスの要件文から，複数の API を推薦する課題に対し，LLM の推論過程を明示した推論手順に従って制御する API 推薦手法を提案した．ソフトウェア工学におけるトップダウン設計の考え方を導入し，要件文に含まれる目的や機能を段階的に具体化しながら API を選定する構造を設計した．また，定義した推論手順に従った事例を作成し，LLM が一貫した推論過程を模倣できるようにした．これにより，従来の汎用的なプロンプトよりも有効であることを示した．本研究の貢献は以下のとおりである．

LLM の推論を導く手順の設計

要件文を直接 API に対応付けるのではなく，サービスの目的を達成するために必要な機能を段階的に具体化する推論手順を設計した．具体的には，要件文の解釈に基づく機能の整理，API カテゴリによる探索範囲の限定，カテゴリ内での API 候補の比較検討を経て，最終的な API を決定する構造を明示的に定義した．これにより，LLM が推論の途中で判断を省略したり，手順を逸脱したりすることを防ぎ，一貫性のある API 推薦を可能にした．

定義した推論手順に基づいた事例の作成

既存データには含まれていない中間的な判断過程については，推論手順との整合性を保つよう補完することで，段階的な判断構造を持つ事例を生成した．これにより，LLM は最終結果のみを模倣するのではなく，要件文から API 推薦に至るまでの推論の流れそのものを学習でき，入力要件の違いに対しても安定した推薦結果が得られることを示した．

今後，実世界において複合サービスを設計および構築する際には，要件文から直接 API を選定するのではなく，本研究で提案したような段階的な推論手順に基づいて API 候補を絞り込むアプローチが有効であると考えられる．特に，複数の API を組み合わせる複合サービスでは，各 API の役割や機能分担を明確にしながら選定することが重要であり，本手法はその設計支援に貢献できると考える．

一方で，本研究では API のカテゴリ情報と説明文を主な知識源として用いており，API の入力と出力，仕様や非機能要件は考慮していない．今後は，これらの情報も推論手順に組み込むことで，より実用的で精緻な API 推薦が可能にな

ると考えられる．また，生成した推論過程の品質評価や，人間の設計者との協調を前提としたインタラクティブな推薦手法への拡張も今後の課題である．

謝辞

修士論文の執筆にあたり，多くの方々から多大なるご指導とご支援を賜りました．ここに深く感謝の意を表します．まず，本研究を進めるにあたり，終始懇切丁寧なご指導と貴重なご助言を賜りました指導教員である村上 陽平先生に，心より感謝申し上げます．研究の方向性から論文執筆に至るまで，常に的確なご指摘をいただき，本研究を完成させることができました．また，本研究におけるプロンプトの添削，ならびに英語の内容梗概の添削をしていただいた Mondheera PITUXCOOSUVARN 先生にも心より感謝申し上げます．さらに，研究活動を通じて多くの励ましをいただいた社会知能研究室の後輩の皆様，ならびに日頃より支えてくださった友人の皆様に感謝いたします．

参考文献

[1] Rong, D., Yao, L., Zheng, Y., Yu, S., Xu, X., M., Wang. X: LLM Enhanced Representation for Cold Start Service Recommendation. In: International Conference on Service-Oriented Computing, pp. 153-167 (2024).

[2] Xu, Z., Gu, Y., and Yao, D.: WARBERT: A Hierarchical BERT-based Model for Web API Recommendation. arXiv preprint arXiv:2509.23175 (2025).

[3] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E.H., Le, Q.V., Zhou, D.: Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In: International Conference on Neural Information Processing Systems, pp. 24824-24837 (2022).

[4] Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A.,Shyam, P., Sastry, G., Askell, A., et al.: Language Models are Few-Shot Learners. In International Conference on Neural Information Processing Systems, pp. 1877-1901 (2020)

[5] Elgazzar, K., Hassan, A.E., Martin, P.: Clustering WSDL Documents to Bootstrap the Discovery of Web Services. In: IEEE International Conference on Web Services, pp.147-154 (2010)

[6] Wu, H., Duan, Y., Yue, K., Zhang, L.: Mashup-oriented Web API Recommendation via Multi-Model Fusion and Multi-Task Learning. IEEE Transactions on Services Computing, 15(6), pp.3330-3343 (2022).

[7] Liu, M., Tu, Z., Xu, H., Xu, X., Wang, Z.: DySR: A Dynamic Graph Neural Network. Based Service Bundle Recommendation Model for Mashup Creation. IEEE Transactions on Services Computing, 16(4), 2592–2605 (2023).

[8] Wang, X., Zhou, P., Wang, Y., Liu, X., Liu, J., Wu, H.: ServiceBERT: A Pre-trained Model for Web Service Tagging and Recommendation. In: International Conference on Service-Oriented Computing, pp. 464–478 (2021).

[9] Gu, Y., Cao, J., Guo, Y., Qian, S., Guan, W.: Plan, Generate and Match: Scientific Workflow Recommendation with Large Language Models. In: International Conference on Service-Oriented Computing, pp. 86–102

(2023).
[10] Qin, S., Zhao, Y., Wu, H., Zhang, L., He, Q.: Harnessing the Power of Large Language Model for Effective Web API Recommendation. IEEE Transactions on Industrial Informatics, 21(7), 5360–5370 (2025).
[11] Feng, S., Cao, B., Xie, Z., Fu, Z., Peng, Z., Kang, G.: LLMSARec: large language model with semantic alignment for Web service. International Journal of Web Information Systems, 21(1), pp. 37–53 (2025).
[12] Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., Iwasawa, Y.: Large Language Models are Zero-Shot Reasoners. In: International Conference on Neural Information Processing Systems, pp.22199–22213 (2022).
[13] Wang, L., Xu, W., Lan, Y., Hu, Z., Lan, Y., Lee, R.K.-W., Lim, E.-P.: Plan-and-Solve Prompting: improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. In: Annual Meeting of the Association for Computational Linguistics, pp. 2609–2634 (2023).
[14] OpenAI.: gpt-oss-120b & gpt-oss-20b Model Card. arXiv preprint arXiv:2508.10925 (2025).
[15] DeepSeek-AI.: DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv preprint arXiv:2501.12948 (2025).
[16] Llama Team, AI@Meta: The Llama 3 Herd of Models. arXiv preprint arXiv:2407.21783 (2024).

付録

A.1 ハルシネーションの詳細

表 10: 各要件文内で起きたカテゴリのハルシネーション数

	Zero-Shot	Zero-Shot CoT	Few-Shot CoT	Plan and solve	Ours
1 個	4	6	8	4	1
2 個	0	0	1	1	0

表 11: 各要件文内で起きた API のハルシネーション数

	Zero-Shot	Zero-Shot CoT	Few-Shot CoT	Plan and solve	Ours
1 個	111	61	54	110	53
2 個	20	12	4	21	1
3 個	6	3	1	5	1
4 個	1	1	0	2	0
5 個	1	0	0	2	0
6 個	1	0	0	0	0
7 個	0	0	0	0	0
8 個	1	0	0	0	0
9 個	1	0	0	0	0
10 個	1	0	0	1	0
>10 個	2	0	0	0	0