論文要約

LLM関連

概要

LLMエージェントSELFGOALは、目標を達成するために、 サブゴールのツリー構造（GOALTREE）を使用し、サブゴールを適応的に分解 環境と対話するために、エージェントは GOALTREEのサブゴールを選択し、分解します。
https://selfgoal-agent.github.io/

手法

SELFGOALは、エージェントが環境と対話しながら目標をサブゴールに分解し、それを基にアクションを生成するためのガイドラインを提供します。 以下に、SELFGOALの3つの主要なモジュールについて説明します

1. **Searchモジュール**: 現在の状態とGOALTREEの既存ノードを基に最も適したサブゴールを選択します。
2. **Decompositionモジュール**: サブゴールをさらに具体的なサブゴールに分解し、GOALTREEを自己成長させます。
3. **Actモジュール**: 選択されたサブゴールをガイドラインとしてLLMにプロンプトを提供し、現在の状態に対するアクションを生成します。

1. Searchモジュール

目的: 現在の状態に基づいて、GOALTREE内の最も適したサブゴールを選択する。

手順:

1.
   - **初期設定:** タスクの高レベルの目標 g0 をGOALTREEのルートノードとして設定します。初期の状態 s0 とアクション a0 を生成し、初期のポリシー πθ に基づいて最初のGOALTREEを生成します。

     T = g_0 ∪ DECOMPOSE(g_0, {a_0, s_0})
2. **状態の更新:** 各タイムステップ t で、エージェントの現在の状態 st を記述し、GOALTREE内の全てのリーフノードを候補サブゴールリストとしてLLMに提供します。
   - **サブゴールの選択:** LLMにプロンプトを提供して、現在の状態に最も適した上位K個のサブゴールを選択させます。選択されたサブゴールは次のアクションのための新しいプロンプトとして使用されます。

     gi,j = SEARCH(T, st)
   - pt+1 = {pt, gi,j}
3.

2. Decompositionモジュール

目的: サブゴールをさらに具体的なサブゴールに分解し、GOALTREEを自己成長させる。

手順:

1.
   - **サブゴールの分解:** 現在の状態とアクションペア {at,st} に基づいて、選択されたサブゴール gi,j をさらに具体的なサブゴールに分解します。

     G = DECOMPOSE(gi,j, {at, st})
   - **重複ノードのフィルタリング**: 新しいサブゴールが既存のサブゴールとテキスト的に類似しているかどうかをコサイン類似度で評価し、重複するノードをフィルタリングします。

BMW Agents - A framework for task automation through multi-agent collaboration BMWエージェント- マルチエージェント協調によるタスク自動化のフレームワーク2024

概要

BMWエージェントは、マルチエージェントで協力して複雑なタスクを自動化するためのフレームワークで計画、実行、検証の　3段階の仕組みを使い業務プロセスを自動化を行います

技術や手法

- 計画（Planning）: ユーザーの指示をシンプルなタスクに分解し、明確な順序で実行。
- 実行（Execution）: 分解されたタスクをエージェントが実行。
- 検証（Verification）: 実行されたタスクが元の指示を満たしているか独立して確認。

エージェントのワークフロー構成要素

1. エージェント (Agent): 特定のタスクを達成するためにLLM呼び出しを行うオブジェクト。
2. エージェントユニット (Agent Unit): 一緒にタスクを解決するエージェントの集合体。
3. マッチャー (Matcher): タスクに適したエージェントユニットを選択する抽象層。
4. エグゼキューター (Executor): エージェントユニットとのすべての操作を調整。
5. ツール (Tools): データベースやAPIなど、タスク完了に必要な外部機能へのアクセス。
6. ツールボックスリファイナー (Toolbox Refiner): タスク実行中にエージェントに提供するツールのセットを絞り込み。
7. コーディネーター (Coordinator): ワークフロー全体の調整とデータフローの計画、実行、検証を実行。

Auto Cherry-Picker: Learning from High-quality Generative Data Driven by Language オートチェリーピッカー：言語に基づく高品質生成データからの学習024

概要

ACPはロンプトを使い、詳細な説明とレイアウトを生成する LLM を使い、テキストから画像へのモデルで複数の画像を生成。生成されたデータは CLISを使い、品質を評価。高品質なデータを生成するため、データフィルタリングを行う。
https://yichengchen24.github.io/projects/autocherrypicker/

技術や手法

1. **自然言語プロンプトによる生成**:
    - 大規模言語モデル(LLM) を使用して、自然言語の概念リストから詳細な説明と合理的なレイアウトを生成。
    - テキストから画像へのモデルを使用して、生成された説明とレイアウトに基づいて複数の画像を生成。
2. **Composite Layout and Image Score (CLIS)**:
    - **CLIS-L**: レイアウトの合理性を評価。
    - **CLIS-I**: 画像の視覚品質とテキスト説明との整合性を評価。
3. **データフィルタリング**:
    - 生成されたデータをCLISを使用して精査し、高品質な訓練データを選別。
4. **応用と実験**:
    - ロングテール分布および不均衡なデータセットでの実験。
    - 合成データを使用して既存モデルのパフォーマンスを向上。

使用用途

- **視覚認識タスク**:
    - セグメンテーション、検出、視覚表現学習の訓練データ生成。
- **マルチモーダル学習**:
    - マルチモーダル視覚質問応答(VQA) の訓練データ生成。
- **不均衡データセットの改善**:
    - ロングテール分布および不均衡データセットに対するデータ生成とパフォーマンス向上。

Identifying Performance-Sensitive Configurations in Software Systems through Code Analysis with LLM Agents LLMエージェントを用いたコード解析によるソフトウェアシステムのパフォーマンス感度の高い設定の特定24

概要

PerfSenseは開発者とパフォーマンスエンジニアの対話をシミュレートし、 prompt chainingやRAGなどのプロンプト技術を使用し3プロセスを経てシステムの速度や効率に影響を及ぼすかを評価し分類します

使用されるプロンプト技術

- **プロンプトチェイン（ Prompt Chaining）**:
    1. PerfAgentがDevAgentに逐次的にタスクを依頼し、複雑なタスクを簡単なサブタスクに分けて実行します。これにより、 PerfAgentは設定に関連するすべての必要なコードを収集し、詳細な分析を行います。
- **RAG（Retrieval-Augmented Generation）**:
    1. PerfAgentは、DevAgentから得られた情報を基に、外部の知識やコンテキストを取り入れて、設定のパフォーマンス感度をより正確に分類します。この技術により、 LLMのコンテキストサイズ制限を克服し、必要な情報をすべて取得できます。

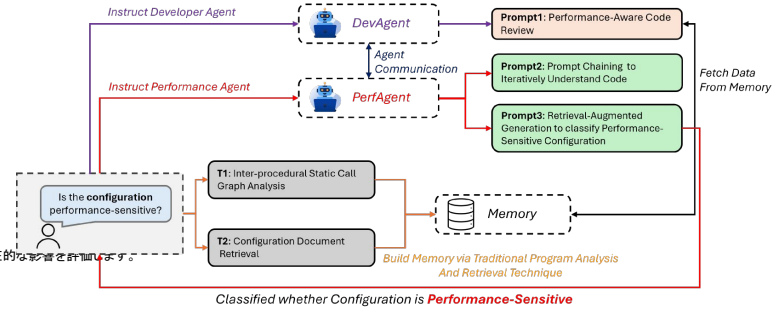
PerfSenseとDevAgentのアルゴリズム説明

PerfSenseの概要

PerfSenseは、パフォーマンスに敏感な設定を特定するための軽量フレームワークです。 LLMエージェントを用いて、開発者（ DevAgent）とパフォーマンスエンジニア（ PerfAgent）の役割をシミュレートし、プロンプトチェインや RAGなどの技術を使って、パフォーマンスに影響を与える設定を効率的に特定します。

アルゴリズムと動作の詳細

1. **設定の取得と初期化**
    - PerfAgentは、分析する設定を受け取ります。これがパフォーマンスに敏感である可能性のある設定です。
    - PerfAgentは、設定に関連するソースコードとドキュメントの取得を DevAgentに依頼します。
2. **設定関連のコードとドキュメントの取得（ DevAgentの役割）**
    - **ソースコードの取得** :
        - DevAgentは、設定に直接アクセスするメソッドを特定します。
        - インタープロシージャルコールグラフを利用して、設定に関連するすべてのメソッドを取得します。
    - **ドキュメントの取得** :
        - DevAgentは、設定に関する公式ドキュメントを取得します。
    - **コードレビューの実施** :
        - DevAgentは、取得したソースコードとドキュメントを基に、コードの機能、設定関連の操作の頻度、および設定がシステムに与える潜在的な影響を評価します。
    - **結果の返却** :
        - DevAgentは、コードレビューの結果を PerfAgentに返します。
3. **パフォーマンス感度の分析と分類（ PerfAgentの役割）**
    - **情報の統合** :
        - PerfAgentは、DevAgentから提供されたソースコード、ドキュメント、およびコードレビューの結果を統合します。
    - **プロンプトチェインの使用** :
        - PerfAgentは、プロンプトチェインを使用して、複雑なタスクを簡単なサブタスクに分け、逐次的に情報を取得し、分析を深めます。
    - **RAGの使用** :
        - PerfAgentは、外部の知識を取り入れて、 LLMのコンテキストサイズ制限を克服し、詳細な分析を行います。
    - **分類の実施** :
        - PerfAgentは、設定がパフォーマンスに敏感かどうかを分類します。この分類は、設定がシステムの速度や効率にどの程度影響を与えるかに基づきます。
4. **結果の出力**
    - PerfSenseは、パフォーマンスに敏感な設定のリストを生成し、パフォーマンスエンジニアに提供します。
    - このリストは、パフォーマンスエンジニアがさらなる調査や最適化のために使用します。



- **動作フローの例**
    1. **設定の受け取り** :

LLMatDesign: Autonomous Materials Discovery with Large Language Models LLMatDesign: 大規模言語モデルによる自律的な材料発見2024

概要

LLMatDesignは、太陽電池材料など、特定のバンドギャップを持つ材料設計を行うフレームワークです。 人間の指示を翻訳し、材料に対する修正を適用し、提供されたツールを使用して結果を評価します。 自己反省を使い次のステップに反省させる k遠出効率的な材料設計ができるようになります

LLMatDesignの技術や手法

LLMatDesignは、材料の化学組成と特性を入力として受け取り、目標とする特性を持つ新しい材料を設計するためのフレームワークです。以下は、そのアルゴリズムの処理順番に沿った詳細な説明です。

1. 初期入力

LLMatDesignは、ユーザーが提供する以下の入力を受け取ります:

- 初期材料の化学組成と特性（例:化学組成 x0、特性値 y0）
- 目標特性値（例:目標特性値 ytarget）

  ytargety_{target}
- 修正履歴（任意、例: M）

2. LLMによる修正提案と仮説生成

大規模言語モデル（ LLM）は、初期材料の組成と特性、および目標特性値に基づいて、次の修正 s_i とその修正の理由となる仮説 hih_ihi を生成します。

- pythonコードをコピーする
- si, hi ← LLM(xi−1, yi−1, ytarget, M)

3. 材料の修正適用

LLMatDesignは、LLMが提案した修正 s_i を材料の組成 x_{i-1} に適用します。修正の種類は以下の 4つです:

- **交換（exchange）**: 材料内の 2つの元素を交換
- **置換（substitute）**: 材料内の 1つの元素を他の元素に置換
- **除去（remove）**: 材料から特定の元素を除去
- **追加（add）**: 材料に新しい元素を追加
  - pythonコードをコピーする
  - x̃i ← perform modification(xi−1, si)

Large Language Models as Evaluators for Scientific Synthesis 科学的総合評価者としての大規模言語モデル2024

概要

LLMで科学文章の要約の品質をどれくらい評価できるかを調査。GPT-4とMistralを使用し、100の研究質問とそれに関連する要約を評価。結果は、LLMが論理的な説明を提供できる一方で、人間の評価と弱い相関が見つかる
先行研究では、ROUGEやBLEUなどの自動評価メトリックが主流でしたが、これらは語彙の一致に依存しており、意味的な情報を考慮していません。

技術や手法

1. **データセット**：CORE-GPTデータセットを使用。これは 100の研究質問とそれに対する GPT-4によって生成された要約を含む。
2. **評価モデル**：GPT-4 TurboとMistral-7Bを使用。これらのモデルに対し、要約の包括性、信頼性、有用性を 0から10で評価するように指示。
3. **評価プロンプト**：タスクの指示、評価基準、回答の形式を含むプロンプトを設計。評価は JSON形式で返却。
4. **評価方法**：人間の評価者との相関を Spearmanのρを用いて分析。

評価プロンプト基準

評価に際しては以下の特性を考慮してください。各特性について 0（非常に悪い）から 10（非常に良い）までの範囲で評価を行い、各評価に対する簡単な理由も提供してください。

1. **包括性 (Comprehensive)**: 質問にどれだけ包括的に答えられているか？
2. **信頼性 (Trust)**: 回答はどれだけ信頼できるか？
3. **有用性 (Utility)**: 回答はどれだけ有用か？

LANE: Logic Alignment of Non-tuning Large Language Models and Online Recommendation Systems for Explainable Reason Generation LANE: 説明可能な理由生成のための非チューニング大規模言語モデルとオンライン推薦システムの論理整合 2024

概要

LANEは非チューニングのLLMをオンライン推薦システムに使用する方法を提案。従来のように項目 IDではなく項目タイトルの埋め込みを使用、CoTにより説明可能な推薦を生成するようにしています

技術や手法

1. セマンティック埋め込みモジュール

目的: ユーザーのインタラクションシーケンスをセマンティックに理解し、推薦の質を向上させる。

手順:

1. **テキストエンコーダの選択** :
    - Sentence-BERTを使用して、項目タイトルを高次元ベクトルに変換する。
2. **項目タイトルの埋め込み** :
    - 全ての項目タイトルをテキストエンコーダに入力し、埋め込み行列　M を生成する。

      MM

    - MMM は、項目数 ∣I∣ と埋め込み次元 d を持つ行列である。

      ∣I∣∣I∣

      dd

A Comparative Study of DSL Code Generation: Fine-Tuning vs. Optimized Retrieval Augmentation DSLコード生成の比較研究 ファインチューニングvs. 最適化されたリトリーバル拡張2024

概要

DSL（ドメイン特化言語）のコード生成を LLMで行うためにRAGの最適化を行ったものとCodexモデルをDSL用にファインチューニングを比較し最適化された RAGモデルは、ファインチューニングされたモデルと同等になった

技術や手法

1. **自然言語プロンプトによる生成**:
    - 大規模言語モデル(LLM) を使用して、自然言語の概念リストから詳細な説明と合理的なレイアウトを生成。
    - テキストから画像へのモデルを使用して、生成された説明とレイアウトに基づいて複数の画像を生成。
2. **Composite Layout and Image Score (CLIS)**:
    - **CLIS-L**: レイアウトの合理性を評価。
    - **CLIS-I**: 画像の視覚品質とテキスト説明との整合性を評価。
3. **データフィルタリング**:
    - 生成されたデータをCLISを使用して精査し、高品質な訓練データを選別。
4. **応用と実験**:
    - ロングテール分布および不均衡なデータセットでの実験。
    - 合成データを使用して既存モデルのパフォーマンスを向上。

使用用途

- **視覚認識タスク**:
    - セグメンテーション、検出、視覚表現学習の訓練データ生成。
- **マルチモーダル学習**:
    - マルチモーダル視覚質問応答(VQA) の訓練データ生成。
- **不均衡データセットの改善**:
    - ロングテール分布および不均衡データセットに対するデータ生成とパフォーマンス向上。

Appendix