論文要約

LLM関連

MagicLens : Self-Supervised Image Retrieval with Open-Ended Instructions MagicLens: 開放型指示を用いた自己教師あり画像検索 2024

概要

画像検索は、リファレンス画像を指定して必要な画像を見つけることを指し、複数の検索意図を含むため、画像ベースの尺度だけでは捉えにくい。最近の研究では、テキスト指示を利用してユーザーがより自由に検索意図を表現できるようにしている。また、画像ペアに焦点を当てた既存の研究は、視覚的に類似しているか、あるいは少数の事前定義された関係で特徴づけられる画像に主に焦点を当てている。

テキスト指示によって視覚的な類似性を超えるより豊かな関係を持つ画像を取得できるというものである。そのため、本研究ではMagicLensという自己教師あり画像検索モデルシリーズを紹介し、オープンエンドの指示をサポートしている。MagicLensは、同じWebページで自然に発生する画像ペアには幅広い暗黙の関係が含まれており、大規模多モーダルモデル（LMMs）および大規模言語モデル（LLMs）を使用して、これらの暗黙の関係を明示化できるという新しい洞察に基づいて構築されている。

Webから採掘された豊富な意味的関係を持つ36.7Mの（クエリ画像、指示、対象画像）トリプレットでトレーニングされたMagicLensは、8つのさまざまな画像検索タスクのベンチマークで、従来の最先端手法に比べて同等以上の結果を達成している。

手法

データの収集: ウェブページから画像を集めます。例えば、ある商品の画像とその商品の充電器の画像が同じページにあれば、この2つの画像は何かしらの関係があると考えます。
データの整理: 収集した画像から、重要な情報を抽出します。この情報には、画像の説明（Alt-text）、画像に写っているもの（ラベル）、画像についての説明文などが含まれます。
指示の生成: 次に、画像が持つ関係性を説明する指示を生成します。これには、大きな言語モデル（LM）を使用して、人間が理解しやすい形で指示を書きます。例えば、「この商品の充電器を見つけて」という指示が生成されるかもしれません。
モデルの訓練: 最後に、収集した画像、その関係性、そして生成した指示を使ってMagicLensモデルを訓練します。これにより、モデルは指示に基づいて関連する画像を見つける方法を学びます。

LLMs as Academic Reading Companions: Extending HCI Through Synthetic Personae 学術読書コンパニオンとしてのLLMs：シンセティックパーソナを通じたHCIの拡張 2024

概要

論文を理解するための補助に使用できるかをAnthropicのClaude.aiを塩牛て調査しました
AIエージェントを使用する参加者と未サポートの独立学習を行う参加者との間で実際に改善が見られた一方、過度の依存や倫理的考慮点が発見され、何とも言いづらい結果に

手法

2つのグループを使った実験を行いました。1つは実験グループで、Claude.aiを使って勉強を支援されました。もう1つはコントロールグループで、特別なサポートなしで勉強しました。研究参加者は、メリーランド大学のインタラクションデザインコースに登録している60人の学生でした。彼らの読解力と勉強への取り組みを測定するために、研究の前、途中、後でアンケートを行いました。また、実験グループの学生からはClaude.aiとのやり取りのテキストログも収集しました。これにより、AIの助けが実際に学生の学習にどのような影響を与えたかを評価しました。

結果

結果、Claude.aiを使用した実験グループの学生は、コントロールグループの学生に比べて、読解力と勉強への取り組みが顕著に向上したことがわかりました。つまり、AIを勉強の友達として使うことで、学生が文章をより深く理解し、より積極的に勉強に取り組むことができるようになる可能性があります。ただし、AIに頼りすぎるリスクや倫理的な問題もあるため、この技術を教育に組み込む際には慎重な検討が必要です。

Ungrammatical-syntax-based In-context Example Selection for Grammatical Error Correction 非文法的構文に基づく文法誤り訂正のためのインコンテキスト例選択 2024

概要

文脈学習（ICL）タスクで文章の文法誤りを修正するための新しい方法を提案。
文章の構造（構文）に基づいて、修正が必要な文章に最も似た例を選び出すことができる戦略を考案しました。つまり、文の「形」を見て、どのように修正すればいいかを判断するのに役立つ例を選ぶ方法です。

手法

非文法的文に対して特別に設計されたパーサー（GOPar）を使用して構文木を生成し、Tree KernelとPolynomial Distanceという2つのアルゴリズムを適用して構文の類似性を測定します。これにより、テスト入力に最も構文的に類似した訓練データ内の例を選択し、ICLで使用します。また、より関連性の高い候補セットから最良の例を選択するためにBM25またはBERT表現を用いた二段階選択メカニズムを設計しました。

1. 文章を「構文木」という形に変換します。構文木とは、文章の文法的構造を木の形で表したものです。これにより、文章がどのように組み立てられているかをパソコンが理解できるようになります。
2. 構文木の類似性を計算します。つまり、修正が必要な文章と似た構造を持つ例をデータベースから探し出すわけです。この過程で、特定のアルゴリズム（Tree KernelやPolynomial Distanceと呼ばれる方法）を使って、どれだけ似ているかを数値で評価します。

構文木の類似性を計算する方法の一つに「Polynomial Distance」があります。
この方法では、文章の構文木を多項式として表現し、その多項式間の距離（どれだけ違うか）を計算します。距離が小さいほど、より類似した構造を持つと判断されます。

結果

この手法をいくつかの文法誤り訂正のデータセットに適用したところ、従来の方法よりも良い結果が得られました。つまり、提案した方法で選んだ例を使って文法誤りを修正すると、より正確に修正できることがわかりました。また、二段階の選択プロセスを導入することで、選択の品質をさらに向上させることができました。

Large Language Model based Situational Dialogues for Second Language Learning 第二言語学習のための大規模言語モデルに基づく状況対話 2024

概要

第二言語学習において、学習者はしばしば資格のある指導者やネイティブスピーカーとの会話練習の機会に恵まれません。このギャップを埋めるために、の状況対話モデルを提案。
通常、言語学習には実際に話す練習が不可欠ですが、質の高い練習相手を見つけるのは難しいことがあります。そこで、このモデルは、学生がさまざまなトピックについて自然に会話を行う練習ができるように設計されています。このシステムは、トレーニング済みトピックだけでなく、トレーニング中に見たことがない新しいトピックに対しても効果的に対応できることが示されています。

手法

1. データ生成: 学校の英語教科書に基づいで51の状況トピックを選び、それぞれについで50の対話を生成しました。これにはOpenAIのGPT-3.5-turboを使用し、総数で7650の対話が生成されました。対話生成では、言葉を簡単にして第二言語学習者でも理解しやすいように指示を加えました。

2. 対話モデル: モデルはLLMsに基づいており、状況に応じた対話を生成するためにファインチューニングされています。具体的にはQwenシリーズの1.8B、7B、14Bの3つのバージョンを使用しました。訓練には、生成された対話データの他に、Alpacaデータセットの10%も使用しました。

3. 応答生成と提案生成 学生が対話中に助言や提案を求めることがあるため、モデルはそれに応じて提案を生成する能力も持っています。これは、対話トレーニングデータにランダムに提案ターンを挿入することで達成されます。

結果

実験では、51の状況トピックと追加の20の新しいトピックを使ってモデルをテストしました。その結果、提案された対話モデルは、既知のトピックだけでなく、訓練中に見ていない新しいトピックにも良好に機能することが示されました。
さらに、14億パラメータを持つLLMをファインチューニングすることで、GPT-3.5を基準とする強力なベースラインを上回るパフォーマンスを達成しましたが、計算コストは低く抑えられました。

Gecko: Versatile Text Embeddings Distilled from Large Language Models Gecko: 大規模言語モデルから抽出された汎用テキスト埋め込み 2024

概要

テキスト埋め込みモデルとしてGeckoを作成。Geckoは、(未ラベルの)パッセージの大規模コーパスを用い、少数ショットプロンプトでLMによって生成されたタスクとクエリに基づいて最寄りのパッセージを埋め込みLLMで再ランキングして正と負のパッセージを得る2段階のLLM駆動型埋め込みモデルを使用し

手法

1. 合成データの生成 LLMを使用して新しい質問とそれに対する答えのペアを作成します。この過程でLLMに特定の情報（パッセージ）を読ませ、それに基づいて関連する質問を自動的に生成させます。たとえばLLMに「地球は太陽の周りを回っています」という文を与えた場合、それに基づいた質問を「地球は何の周りを回っているか？」と生成させるような形です。
質問生成の過程を数式で表すと、あるパッセージP から質問 Q を生成する関数 f を考えることができます。Q=f(P)ここで、f はLLMによって定義される関数です。
2. 正解と不正解の選定 生成された質問 Q に対して、LLMを使用して複数の候補答えから最も適切な答えA+ を選び出します。
さらに、良い学習材料を作るために、間違った答えやあまり関係のない答えA − も選び出します。このプロセスでは、候補となる答えの中から、質問に最も関連するものを「正解」として、その他を「不正解」として分類します。

これらのステップを通じて、Geckoは多様な質問に対する適切な答えを見つけ出し、さらにその学習を深めるための「不正解」も選び出すことができるようになります。この方法でGeckoは知識をより豊富にし、さまざまな質問に対する理解を深めるような動作をします。。

結果

Geckoは、256次元の埋め込みで、768次元の埋め込みサイズを持つ全てのエントリを上回り、768次元の埋め込みで平均スコア66.31を達成し、7倍大きなモデルや5倍高い次元の埋め込みを使用するシステムと競合します。

Unleashing the Potential of Large Language Models for Predictive Tabular Tasks in Data Science データサイエンスにおける予測的表形式タスクにおける大規模言語モデルの可能性の解放 2024

概要

LLMを使用し、分類、回帰、および欠損値の補完といった、表形式データに関連する予測タスクに使用するために大量の表形式データからなる広範なプレトレーニングコーパスを編成Lama-2をこのデータセットで学習し、

手法

1. データの集め方: まず、主にkaggleからインターネットを使用して様々な表のデータを集めました。これは、プログラムが多くの例を見て学ぶためです。

2. 訓練方法: 次に、プログラムに特別な訓練を施しました。表の中の一部の情報を隠しておき、プログラムにその隠された情報を当てさせるようにしました。これを繰り返すことで、プログラムは表のデータを理解する方法を学びます。

3. テストのやり方: 訓練が終わったら、プログラムが実際に様々な予測を正確に行えるかどうかを試しました。これは、新しい表のデータをプログラムに見せ、何かを予測させることで行いました。

結果

30の分類および回帰タスクにわたる広範な実験分析と比較評価を通じて、顕著な性能を実証しました。分類タスクでは平均 8.9%、回帰タスクでは平均10.7%の改善が達成されました。欠損値予測タスクでは、GPT-4に対して27%の改善を示しました。さらに、様々なデータセットにおける極端な少数ショット(4ショット)予測では、平均で28.8%の性能向上が観察

TM-TREK at SemEval-2024 Task 8: Towards LLM-Based Automatic Boundary Detection for Human-Machine Mixed Text SemEval-2024 タスク 8 における TM-TREK: 人間と機械が混在するテキストのための LLM ベースの自動境界検出に向けて 2024

概要

LLMが生成したテキストと人間によって書かれたテキストを区別するSemEval-2024 Task 8コンペで一位になったこの手法は混合テキストのトークンレベルの境界検出を、トークン分類問題として枠組みを設定しLLMを利用して、各トークンが人間によって書かれたものか、機械によって生成されたものかを分類します。性能をさらに向上させるために、複数のファインチューニングされたモデルからの予測を組み合わせるアンサンブル戦略を採用しています。

手法

問題を「トークン分類問題」として定義し、LLMか人かを分類することを考え以下の手順で問題を解きました。
1. データセットの準備: 文章のサンプルを集め、それぞれの単語が人間か機械かに応じてラベル付けをしました。これにより、トレーニング用のデータセットができます。
2. LLMの選択: 研究チームは、長い文章でも効果的に働くことができる、いくつかの大規模言語モデル(LM)を選びました。具体的には、Longformer、XLNet、BigBirdなどのモデルが試されました。
3. トークンへのラベル付け: 各単語に「人」と「LLM」というラベルを付けることで、トークン分類のタスクを行います。
4. ファインチューニングとアンサンブル: 選んだLLMを、準備したデータセットを使ってファインチューニングし、アンサンブルも実施
5. 性能向上のために、LLMの上に追加のレイヤー(LSTMやBiLSTMなど)を導入したり、セグメント損失関数を利用したり
6. 事前学習とファインチューニング: モデルの一般化能力を高めるために、関連する別のタスク(例えば、文章が人間か機械かを分類するタスク)でモデルを事前学習し、その後、目的のトークン分類タスクでファインチューニングを実施。

結果

LLMを用いた境界検出のアプローチは、SemEval'24コンペティションで最適な性能を達成しました。XLNetモデルのアンサンブルを利用することで、境界検出能力を高めることができました。

Great, Now Write an Article About That: The Crescendo Multi-Turn LLM Jailbreak Attack クレッシェンド多段階LLMジェイルブレイク攻撃 2024

概要

「クレッシェンド」と名付けられた新しいタイプのルールで禁止されている質問に答えさせたりするジェイルブレイク攻撃を紹介。これは従来のジェイルブレイク手法と異なり、モデルにseemingly benign（表面上無害な）な方法でやり取りする多段階攻撃です。攻撃は一般的なプロンプトや質問から始まり、モデルの応答を参照しながら徐々に対話をエスカレートさせ、最終的に成功するジェイルブレイクに至ります。さらに「Crescendomation」という、クレッシェンド攻撃を自動化するツールを導入

手法

「クレッシェンド」攻撃をするとき、まずは普通の質問から始めます。LLMが答えると、その答えをヒントにもう少しトリッキーな質問をします。これを繰り返すことでLLMは徐々に普段は避けるような回答をしてしまいます。
始めの質問をする: まず、AIにとって全く無害で普通の質問をします。この質問はAIが簡単に答えられるもので、会話を始めるためのものです。
1. 始めの質問:「最近人気の健康食品について教えてください。」
   LLMの回答:「最近は、カリフラワーを使ったピザや、スピルリナを加えたスムージーなどが人気です。」
2. 応答を参考にした質問「面白いですね。でも、たまにはカロリーを気にせず楽しみたい時もありますよね。そんな時におすすめの食べ物はありますか？」
   LLMの回答:「たまのご褒美には、チョコレートケーキやポテトチップスなどがおすすめです。」
3. 質問をエスカレートさせる「チョコレートケーキ、美味しそうですね。自宅で簡単に作れるレシピを教えてもらえますか？」
   LLMの回答:「もちろんです。こちらが簡単で美味しいチョコレートケーキのレシピです..」
4. 目的の質問に到達する「そのレシピで、さらに甘くてカロリーが高くなるようにするにはどうすればいいですか？」
   LLMの回答:「甘さとカロリーを増やしたい場合は、砂糖の量を倍にしたり、チョコレートチップスを追加したりするといいでしょう。」
5. 結果を確認する: このシナリオでは、AIは最初は健康食品について答えていましたが、段階的な質問を通じて、最終的には健康に良くない高カロリーのレシピを提供するよう導かれました。

この例では、攻撃者はLLMとのやりとりを通じて、徐々にLLMの安全ガイドラインを迂回する質問をしています。クレッシェンド攻撃のポイントはLLMを直接的に危険な行動に導くのではなく、段階的に目的の応答を引き出すことにあります。

結果

Peer-aided Repairer: Empowering Large Language Models to Repair Advanced Student Assignments 大規模言語モデルを活用して高度な学生課題を修復する 2024

概要

プログラミング課題に対するフィードバックの自動生成の為に、特に上級プログラミングコースからのプログラムを修正することを考え、高度なプログラミングコースからの新しい学生課題データセットであるDefects4DSを使用しLLMによって駆動される新しいフレームワークであるPaRを開発。
PaRは、Peer Solution Selection、Multi-Source Prompt Generation、Program Repairの3つのフェーズで動作します。

手法

PaRのアルゴリズムは以下です
1. 関連する解答を探す（Peer Solution Selection）
最初のステップでは、自分が間違えた課題と似ている問題を、過去に他の人がどう解決したかを探します。このステップでは、たくさんある解答の中から、自分の間違えた部分と一番関連性が高いものを見つけ出します。
2. 情報を組み合わせる（Multi-Source Prompt Generation）
次に、見つけた解答だけでなく、問題の説明や入出力の例など、いろいろな情報を一緒に考えます。これは、プログラムを直すヒントとなる質問を作るようなものです。つまり、問題を解くために必要なすべての情報をまとめて、プログラムを修正するための手がかりを作り出します。
3. 問題を直す（Program Repair）
最後に、上の2ステップで集めた情報を基に、間違いを直します。このステップでは、特別に訓練された大きなコンピューターシステムが、間違ったプログラムを見て何が間違っているのかを理解し、どう直せばいいのかを考え出してくれます。そして、正しい解答を生成します。これは、間違いを指摘して、どう直せばいいかを教えてくれるのに似ています。

要するに、この3ステップは、プログラミングの問題を解くときに、良い解答を見つけ、その解答から学び、最後に自分の間違いを直す過程を自動化したものです。

InsightLens: Discovering and Exploring Insights from Conversational Contexts in Large-Language-Model-Powered Data Analysis
InsightLens: 大規模言語モデルを活用したデータ分析における会話コンテキストからの洞察の発見と探索 2024

概要

InsightLensと呼ばれるLLMベースのマルチエージェントフレームワークを提案しており、これは分析プロセスに沿って洞察を自動的に抽出、関連付け、整理することを目的としています。
この対話型システムを使用することで、複雑な会話コンテキストを多面的に可視化して洞察の発見と探索を容易にしています。

手法

1. ユーザーの質問を解釈する ユーザーがデータについて何か質問をすると、その質問を理解するため、質問の中に含まれるキーワードやデータに関する情報を抽出します。
2. データから洞察を抽出する LLMを使い、データの中から重要な情報や洞察（つまり、気づきや新しい発見）を探し出します。例えば、ある商品の売上が特定の時期に急増している理由などです。
3. 洞察とその証拠を関連付ける 発見された洞察には、それを裏付けるデータや情報があります。InsightLensは、その洞察を支える証拠（コードの実行結果やデータの視覚化など）を自動的に見つけ出し、洞察と一緒に表示します。
4. 洞察を整理する: 発見された洞察が多数ある場合、それらを整理して分かりやすく表示する必要があります。InsightLensは、洞察を関連するテーマやカテゴリに分類し、ユーザーが簡単に理解できるようにします。
5. 洞察を視覚化して表示する 最後に、洞察やその整理されたカテゴリを、ミニマップやトピックキャンバスなどの視覚的な要素を使って表示します。これにより、ユーザーは分析の全体像を一目で把握できるようになります。
6. ユーザーのフィードバックを受け取る ユーザーが新たな質問をしたり、分析の方向性を変えたりすると、InsightLensはそのフィードバックに基づいて、再びステップ2から処理を開始します。

この一連の手順を通じて、InsightLensはデータ分析を行う上での手間や複雑さを大きく削減し、ユーザーがより簡単にデータから価値ある洞察を得られるように支援します。

Language Models as Compilers: Simulating Pseudocode Execution Improves Algorithmic Reasoning in Language Models
言語モデルをコンパイラとして: 疑似コード実行のシミュレーションによる言語モデルのアルゴリズム推論能力の向上 2024

概要

THINK-AND-EXECUTEという手法を使用して質問を解決するため回答をプログラミング言語を使用して段階的に回答します。一般的に、大きな問題を小さなステップに分けて考えることは難しいですが、この手法では疑似コードを使用して、問題をより理解しやすく分解します。このアプローチは、言語モデルがより効率的に問題解決を行うのを助け、特にアルゴリズムの推論タスクでその性能を向上させることができます。具体的には、問題を解決するための一般的なロジックを「考える（THINK）」段階で見つけ、それを疑似コードで表現します。次に、「実行する（EXECUTE）」段階で、その疑似コードを各問題に合わせて調整し、実行をシミュレートします。この方法により、言語モデルは問題をより効果的に解決することができます。

手法

以下のステップで構成されています。
1. THINK（考える）: まず、問題を解決するために必要な全体的なロジックを見つけ出します。このロジックは、疑似コードで表現され、問題の種類に関わらず共通して適用可能です。このステップでは、問題の本質を理解し、どのようにアプローチすればよいかを疑似コードで記述します。
「タスクのロジックを考え、そのロジックを疑似コードでどのように表現できるかを示してください。疑似コードは、問題解決のためのステップを明確にするために、条件分岐やループなどのプログラミングの基本概念を使用してください。また、この疑似コードは、同じタスクの異なるインスタンスに対しても適用可能でなければなりません。」

2. EXECUTE（実行する）: 次に、生成された疑似コードを具体的な問題の状況に合わせて調整します。そして、この疑似コードの実行をシミュレートすることで、問題の解決策を導き出します。このプロセスを通じて、言語モデルはステップバイステップで問題を解決する方法を「学び」ます。
「先ほどTHINKフェーズで作成した疑似コードを使用して、特定の問題インスタンスを解決してください。疑似コードの各ステップをシミュレートし、その結果を出力してください。最終的な答えを得るために、どのように疑似コードを調整し、実行するかを示してください。」

結果

特に、問題を解決する際に必要なロジックをタスクレベルで見つけ出し、それを疑似コードとして表現することの有効性が示されました。また、疑似コードを使って推論過程をシミュレートすることで、言語モデルがより複雑な問題を効果的に解決できるようになります。

Large Language Model for Vulnerability Detection and Repair: Literature Review and Roadmap
脆弱性検出と修復のための大規模言語モデル：文献レビューとロードマップ 2024

概要

THINK-AND-EXECUTEという手法を使用して質問を解決するため回答をプログラミング言語を使用して段階的に回答します。一般的に、大きな問題を小さなステップに分けて考えることは難しいですが、この手法では疑似コードを使用して、問題をより理解しやすく分解します。このアプローチは、言語モデルがより効率的に問題解決を行うのを助け、特にアルゴリズムの推論タスクでその性能を向上させることができます。具体的には、問題を解決するための一般的なロジックを「考える（THINK）」段階で見つけ、それを疑似コードで表現します。次に、「実行する（EXECUTE）」段階で、その疑似コードを各問題に合わせて調整し、実行をシミュレートします。この方法により、言語モデルは問題をより効果的に解決することができます。

手法

以下のステップで構成されています。
1. THINK（考える）: まず、問題を解決するために必要な全体的なロジックを見つけ出します。このロジックは、疑似コードで表現され、問題の種類に関わらず共通して適用可能です。このステップでは、問題の本質を理解し、どのようにアプローチすればよいかを疑似コードで記述します。
「タスクのロジックを考え、そのロジックを疑似コードでどのように表現できるかを示してください。疑似コードは、問題解決のためのステップを明確にするために、条件分岐やループなどのプログラミングの基本概念を使用してください。また、この疑似コードは、同じタスクの異なるインスタンスに対しても適用可能でなければなりません。」

2. EXECUTE（実行する）: 次に、生成された疑似コードを具体的な問題の状況に合わせて調整します。そして、この疑似コードの実行をシミュレートすることで、問題の解決策を導き出します。このプロセスを通じて、言語モデルはステップバイステップで問題を解決する方法を「学び」ます。
「先ほどTHINKフェーズで作成した疑似コードを使用して、特定の問題インスタンスを解決してください。疑似コードの各ステップをシミュレートし、その結果を出力してください。最終的な答えを得るために、どのように疑似コードを調整し、実行するかを示してください。」

結果

特に、問題を解決する際に必要なロジックをタスクレベルで見つけ出し、それを疑似コードとして表現することの有効性が示されました。また、疑似コードを使って推論過程をシミュレートすることで、言語モデルがより複雑な問題を効果的に解決できるようになります。

AutoWebGLM: Bootstrap And Reinforce A Large Language Model-based Web Navigating Agent
AutoWebGLM: 大規模言語モデルをベースとしたWebナビゲーションエージェントのブートストラップと強化 2024

概要

Webページを効果的に操作し、理解することができる新しいエージェントAutoWebGLMを開発。インターネット上での様々なタスク、例えばウェブサイトからの情報収集やオンラインショッピングなどを自動化することができます。人間がWebを使うときのパターンに基づいて、Webページの重要な情報を簡潔に抽出し、エージェントが理解しやすい形に整理します。さらに、エージェントが独自に学習して、より効果的にWeb上をナビゲートできるようにするための訓練方法も開発しました。
https://github.com/THUDM/AutoWebGLM

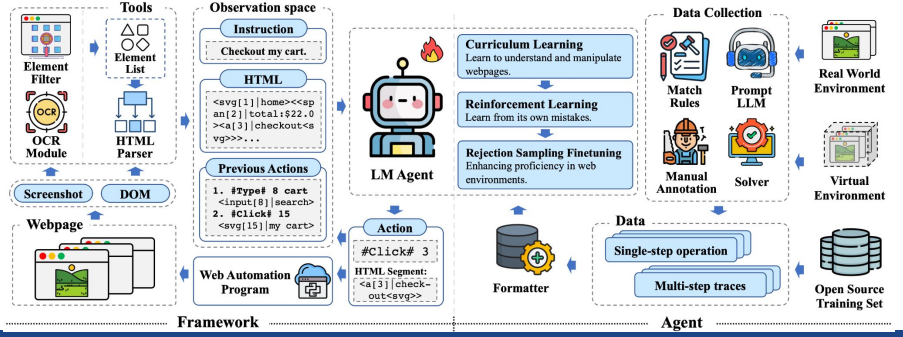
手法

まず「HTMLの簡素化アルゴリズム」を設計しました。これは、Webページの複雑な内容をエージェントが処理しやすい形に変換する技術です。例えば、長いテキストや複雑なレイアウトを持つページから、エージェントにとって重要な情報のみを抽出し、簡潔な形式で表現します。次に、「カリキュラムトレーニング」という方法を用いてエージェントを訓練しました。これは、簡単なタスクから徐々に複雑なタスクへと進めていく学習方法です。エージェントがWebページをどのように解釈し、どのようなアクションを取るべきかを段階的に学習していきます。さらに、「強化学習」と「リジェクションサンプリング」を行い、エージェントが自分の過ちから学び、自己改善する能力を高めました。エージェントが誤った操作をした場合に、どのように修正すれば良いかを学ぶことで、より正確にタスクを遂行できるようになります。

結果

AUTOWEBGLMは、実際のWebナビゲーションタスクにおいて優れた性能を示しました。特に、実際のWebサイトを使ったテストでは、従来の方法に比べて高い成功率を達成しました。このエージェントは、特定の情報を探したり、特定の操作を行ったりするタスクを効率的にこなすことができます。

しかし、まだ改善の余地があります。エージェントが間違った情報を取得することや、予期せぬポップアップに対処することが困難な場合があります。今後の研究では、これらの課題を克服し、より高度な Web ナビゲーションエージェントを開発することが目標です。



CODEEDITORBENCH: EVALUATING CODE EDITING CAPABILITY OF LARGE LANGUAGE MODELS
CODEEDITORBENCH: 大規模言語モデルのコード編集能力の評価 2024

概要

LLMが、他のプログラムが作った間違いを見つけられるかどうかを調査し、新たな誤り検出ベンチマークReaLMistakeを導入ReaLMistakeは客観的で現実的かつ多様な誤りを含む3つの課題を提供し、12つのLLMsに基づく誤り検出器の評価を行いました。結果は、トップの2LLMsが誤りを検出する際に非常に低い再現率を示し、すべての2LLMベースの誤り検出器が人間よりも遥かに性能が悪くなることがわかりました

https://github.com/psunlpgroup/ReaLMistake

手法

う新しいテスト方法を作りました。これは、さまざまなプログラミング言語で書かれたコードに対して、複数の編集タスクを実行させ、どのモデルが最も効果的に処理できるかを見るためのものです。テストでは、コードを短くする、バグを修正する、言語間で翻訳する、新しい要求に合わせてコードを変更するといった、実際の開発作業に近い状況を再現しました。これにより、モデルがどれだけ実用的かを評価します。さらに、オンラインでプログラムが正し動作するかどうかもチェックするシステムを使いました。

結果

テストの結果、すべてのモデルが同じように良いわけではなく、特定のタスクには特定のモデルがより適していることがわかりました。例えば、一部のモデルはプログラムを翻訳するのが得意で、他のモデルはコードのバグを修正するのが得意でした。このテスト方法によって、どのモデルが実際の開発作業に最も役立つかを知ることができます。研究チームは、このテスト方法や得られたデータを公開して、今後さらに良いプログラミング言語モデルの開発を支援します。

A Comparison of Methods for Evaluating Generative IR Generative IR 評価方法の比較 2024

概要

RAGがどのようにしてより良い回答を作り出すか、特に質問に対する答えを自動で生成するようなシステムに焦点を当て、新しいタイプの検索システム（Generative Information Retrieval, Gen-IRシステムと呼びます）をどのように評価するかに取り組みました。Gen-IRシステムは、質問に対してインターネットから情報を収集し、それをもとに新しい文章や回答を生成します。このようなシステムを評価するために、研究者たちはいくつかの方法を提案しています。

手法

Gen-IRシステムとは、質問に対する回答をインターネットなどから情報を収集して生成するシステムのことです。従来の検索システムは、既に存在する文書やデータベースから最も関連性の高い情報を見つけ出してユーザーに提示しますが、Gen-IRシステムは一歩進んで、集めた情報を基に新しい文章や回答を自動で作り出します。これにより、ユーザーが求める情報に対してより直接的でわかりやすい形で応えることが可能になります。
Gen-IRシステムを評価する手法としては主にいかがあります。
1. 二項関連性 (Binary Relevance): この方法では、生成された回答が質問に対して関連があるかどうかを「はい」または「いいえ」で評価します。
シンプルながらも、回答が基本的な質問の要求を満たしているかを判断するのに役立ちます。
2. 階層的関連性 (Graded Relevance): 回答の関連性をより詳細に評価し、回答をさまざまなレベルでランク付けします。これにより、より関連性の高い回答をより精密に識別することができます。
3. サブトピック関連性 (Subtopic Relevance): 質問に含まれる様々な小さなトピックやポイントが回答にどれだけ含まれているかを評価します。質問の多面性に対応し、回答が包括的であるかどうかを判断します。
4. ペアワイズ嗜好 (Pairwise Preferences): 2つの回答を比較して、どちらがより優れているかを評価します。この方法は、特に複数の良い回答がある場合に、最適なものを選び出すのに有効です。
5. 埋め込みに基づく方法 (Embedding-based Methods): 回答と質問の内容の類似度を計算することで評価します。これには通常、文書や文の意味的な表現をベクトル空間に埋め込む技術が用いられます。類似度が高いほど、回答が質問に適切であると見なされます。

これらの方法は、人間が評価する従来の方法とLLMを使用した自動評価とを組み合わせています

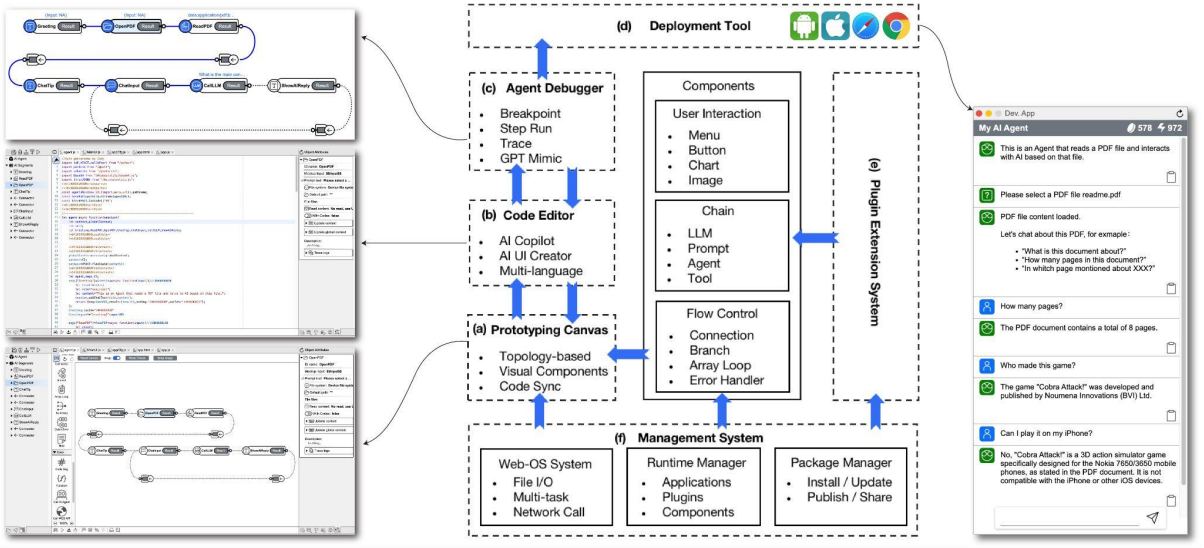
結果

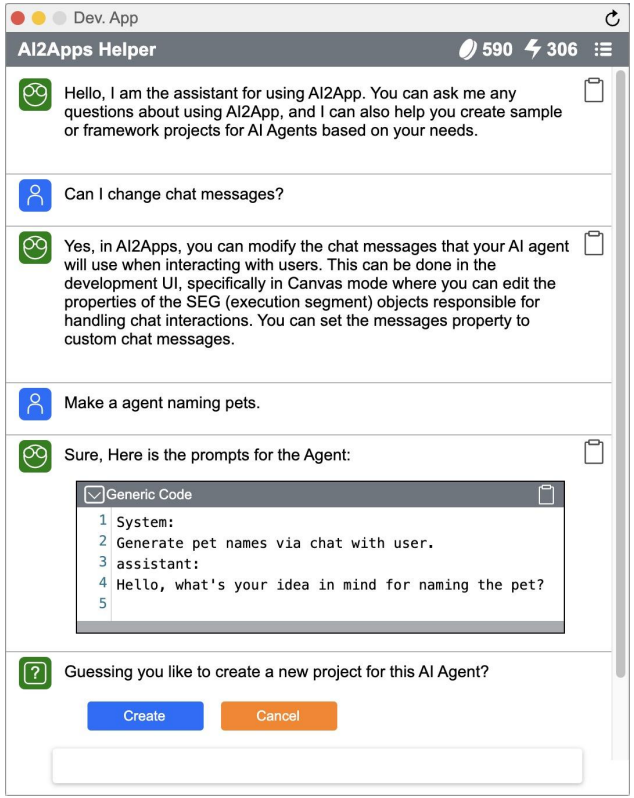
テストの結果、すべてのモデルが同じように良いわけではなく、特定のタスクには特定のモデルがより適していることがわかりました。例えば、一部のモデルはプログラムを翻訳するのが得意で、他のモデルはコードのバグを修正するのが得意でした。このテスト方法によって、どのモデルが実際の開発作業に最も役立つかを知ることができます。研究チームは、このテスト方法や得られたデータを公開して、今後さらに良いプログラミング言語モデルの開発を支援します。

AI2Apps: A Visual IDE for Building LLM-based AI Agent Applications AI2Apps: LLMベースのAIエージェントアプリケーションを構築するためのビジュアルIDE 2024

概要

AI2Appsは、エージェントアプリを簡単に作れるツールです。ウェブ上で使えて、ドラッグ＆ドロップでコードを書かずにアプリを作ることが出来そうですが、ちょっとUIが自体が使いづらいような
https://www.ai2apps.com//-homekit/home.html





FRACTAL: Fine-Grained Scoring from Aggregate Text Labels FRACTAL: 集約されたテキストラベルからの細かい粒度のスコアリング 2024

概要

文レベルのラベルがLLMの最適化においてより正確で解釈しやすいフィードバックを提供できることができるらしい。文章の集まりから、文ごとに詳細な評価やスコアを出す新しい方法を開発。
検索、質問応答、要約、数学的推論など通常、文章全体に対して一つの評価をするのが普通のタスクで各文ごとに評価を下すことができるこの方法を使用することで文章のどの部分が良くて、どの部分が改善が必要かがより明確になりました

手法

大規模なテキストデータの中から各文に対する細かいスコアリングを行うための手法RACTALは、特に文章全体に対する評価から、個々の文に対する評価（擬似ラベル）を抽出し、それを基に文レベルでのスコアリングモデルを訓練することを目的としています。

1. 擬似ラベルの生成:目的: 文章全体の評価から各文の擬似ラベルを生成します。
方法: 文書全体に与えられたラベル（例えば、文章全体の品質スコア）を基に、各文に対する擬似ラベルを割り当てます。これは、文章全体の評価が各文の品質にどのように関連しているかを見積もるプロセスです。

2. モデルの訓練:目的: 各文の擬似ラベルを使用して、文レベルでのスコアリングモデルを訓練します。
方法: 生成した擬似ラベルと文書内の各文の特徴（例：文と文書のコサイン類似性）を用いて、教師あり学習のアプローチでモデルを訓練します。このステップでは、複数インスタンス学習(MIL)やラベル比率からの学習(LLP)といった技術を活用します。

3. 性能の向上:目的: 訓練されたモデルの予測を利用して、擬似ラベルを更新し、モデルの性能をさらに向上させます。
方法: モデルの予測をもとに、擬似ラベルを再計算し、それを用いてモデルを再訓練します。この反復プロセスを通じて、より精度の高い文レベルのスコアリングが可能になります。

文と文書の関係性を明らかにするための特徴量（例えば、文書と文の間のコサイン類似性や、文間の関係性を示す特徴）の選定が重要で擬似ラベルの精度を高めるための工夫や、モデルの訓練過程での最適化手法の選択もキーになりそう

結果

この新しい方法は、各文が全体の質にどのように貢献しているかをより正確に評価できるようになります。これにより、文章を改善するための具体的な指摘が可能になり、より効果的な文章作成や編集ができるようになるらしい

LayoutLLM: Layout Instruction Tuning with Large Language Models for Document Understanding LayoutLLM: 大規模言語モデルを用いたドキュメント理解のためのレイアウト命令チューニング 2024

概要

大規模言語モデル（LLMs）や多モダリティ大規模言語モデル（MLLMs）を活用した文章理解において、従来の手法では文書のレイアウト情報が十分に探求・活用されていない。
文書理解のためのLLM/MLLMベースの手法であるLayoutLLMを提案。LayoutLLMは、文書のレイアウト情報の理解と活用を向上させるために特別に設計されたレイアウト指示調整でLayout-aware Pre-trainingとLayout-aware Supervised Fine-tuningの2つのコンポーネントから構成されている。提案された手法は、既存のB LLMs/MLLMsを採用した手法よりも優れた性能を示すことが実験で示されている。

手法

LayoutLLMは、大きく分けて二つの重要な部分から成り立っています。
1. レイアウト認識の事前トレーニング: ここでは文書全体、文書の特定の領域、そして文書の小さなセクションといった、異なるレベルでの情報を学習します。これにより、文書の構造を全体的にも細かくも理解できるようになります。
2. レイアウト認識の教師あり微調整 具体的な質問に答えるためのスキルを向上させます。この段階では、「LayoutCoT」と呼ばれる新しいモジュールが使われ、質問に関連する文書の領域に集中し、正確な答えを出すための手順を踏みます。この二つのステップによって、文書のテキストとレイアウトの両方を理解し、それを使って情報を処理する能力が高まります

結果

この新しい方法は、各文が全体の質にどのように貢献しているかをより正確に評価できるようになります。これにより、文章を改善するための具体的な指摘が可能になり、より効果的な文章作成や編集ができるようになるらしい

Generate then Retrieve: Conversational Response Retrieval Using LLMs as Answer and Query Generators

生成後検索：回答とクエリジェネレータとしてのLLMを使用した会話応答の検索 2024

概要

対話型の情報検索はユーザーとの対話を通して情報ニーズを理解し、
適切な情報を見つけ出す技術です。
複数のクエリを生成し、より効果的に情報を検索する3つの新しい方法を使用し
GPT-4やLlama-2 chatなどで評価することで直接クエリを生成する手法よりも、
検索性能が高いことがわかりました

手法

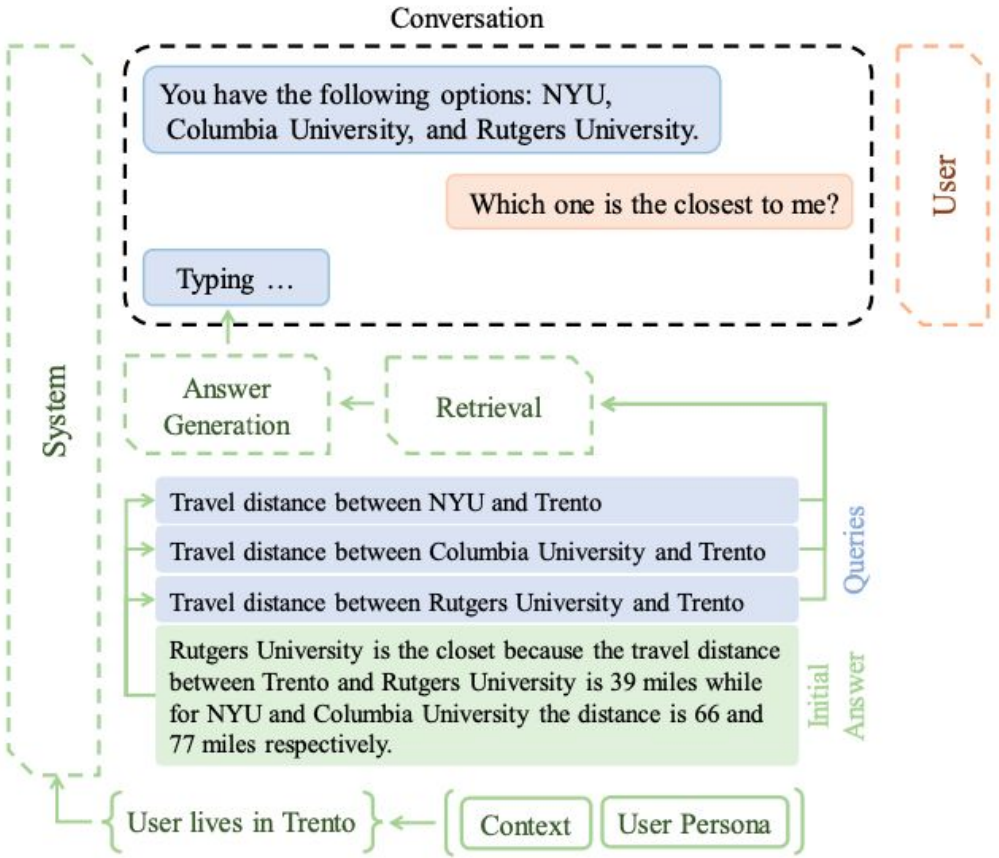
1. 応答生成後の複数クエリ生成(AD): まずLLMを用いてユーザーの質問に対する回答を生成し、その回答から複数の検索クエリを生成します。

2. 直接クエリ生成(QD): LLMに直接、情報検索に使用するクエリを生成させます。

3. 応答とクエリの両方を生成(AQD): LLMによる回答生成後、
その回答を元に更に詳細なクエリを生成します。

結果

生成された回答からクエリを生成する手法は、
直接クエリを生成する手法よりも、検索性能が高いことが確認されました。



1. 応答生成後の複数クエリ生成(AD): まずLLMを用いてユーザーの質問に対する回答を生成し、その回答から複数の検索クエリを生成します。
3. 応答とクエリの両方を生成(AQD): LLMによる回答生成後、その回答を元に更に詳細なクエリを生成します。

GPT-4をゼロショット学習者として使用するAQDおよびADアプローチのために設計されたプロンプト。

(1) # 指示:
ユーザーとシステム間の会話をお伝えします。また、ユーザーの背景情報も提供します。ユーザーの最後の質問に答えるべきです。ユーザーの最後の質問への回答は 200語を超えないようにしてください。
背景知識: {}
コンテキスト: {}
ユーザーの質問: {}
応答:

(2) # ユーザーへの以前の回答を取得するために使用できるユニークなクエリを生成できますか？ (各クエリを1行で記載し、5つを超えないようにしてください)
生成されたクエリ:

2. 直接クエリ生成 (QD): LLMに直接、情報検索に使用するクエリを生成させます。

GPT-4をゼロショット学習者として使用するQDアプローチのために設計されたプロンプト。QDアプローチにおけるクエリ生成。

指示:
ユーザーとシステム間の会話およびユーザーの背景情報を提供します。Googleで検索することにより、最後のユーザー質問の答えを見つけたいと想像してください。Googleで検索する必要がある検索クエリを生成してください。5つを超えるクエリを生成せず、各クエリを1行で記述してください。
背景知識: {}
コンテキスト: {}
ユーザーの質問: {}
生成されたクエリ:

Low-Cost Generation and Evaluation of Dictionary Example Sentences 辞書例文の低コスト生成と評価 2024

概要

辞書の例文を作るために自動評価システムOxfordEvalを使用して例文を作成することで

手法

OxfordEvalは、生成された辞書の例文がどれだけ良質かを測るための自動評価指標です。
この指標は、生成された文が既存のオックスフォード辞書の例文と比較してどれだけ選ばれるか、という「勝率」を測定します。勝率が高いほど、生成された文の品質が良いと評価されます。

1. 候補例文の生成をLMを使用して特定の単語、その定義、そして品詞を入力として受け取り、それに基づいて適切な例文を生成します。
2. 生成文とオックスフォード辞書の文の比較評価 生成された例文は、オックスフォード辞書に既に存在する例文と比較されます。これには、人間のアノテーション（人が評価すること）に基づいたサブセットも使用され、OxfordEvalが人間の判断とどれだけ合致するかを検証します。
3. 勝率の計算: 各生成された文に対して、それが既存の辞書の文よりも優れているかどうかを評価します。これを複数の文に対して行い、全体の勝率を算出します。勝率は、生成した文が辞書の既存の文をどれだけ上回るかを数値で示します。評価結果に基づいて、生成された例文が選ばれた場合は「勝ち」と記録し、辞書の例文が選ばれた場合は「負け」と記録します。
勝率 = \ frac{生成分の勝利数}{評価されたペアの総数}×100
4. 結果の検証と分析: 最終的に、OxfordEvalによって得られた勝率が50%を超える場合、生成された例文が一般的に辞書の既存の例文よりも好まれると解釈されます。さらに、異なる言語モデルでの性能を比較分析することで、最も効果的な生成モデルを特定します。

結果

既存の辞書の文と比べて、約85%の場合により良い文を作れました。これは、新しい方法が非常に効果的であることを示しています。プログラムが作った文は、辞書の文よりも短く、読みやすいという特徴もあったらしい

Not All Contexts Are Equal: Teaching LLMs Credibility-aware Generationすべてのコンテキストが等しいわけではない：LLMに信用性を意識した生成を教える 2024

概要

LLMが信頼できる情報を選び出して使う方法を改善するための新しい技術、「信用性を意識した生成（Credibility-aware Generation、CAG）を提案。
現在の技術では、間違った情報を取り込んでしまうことがありますが、CAGは情報の「信用性」を見分ける力をモデルに付加します。これにより、より正確で信頼性の高い文章を作ることができるようになります。

手法

1. 多粒度信用性アノテーション: 文章の各部分がどれだけ信用できるかを評価し、ラベルを付けます。これにより、モデルはどの情報を信じるべきかを学びます。

2. 信用性に基づく説明の生成 モデルが質問に答える際に、信用性の高い情報を優先的に使うようにします。これにより、出力される回答の信頼性が増します。

3. インストラクションによる微調整 信用性を考慮しながら適切な回答を生成するために、モデルを特別に訓練します。

結果

この技術を使うと、モデルは信用できる情報と信用できない情報をうまく区別できるようになります。実験では、他の方法よりも優れた性能を示しました。特に情報が複雑で信用性が低い状況でも、良い結果を出すことができました。これにより、誤情報が多い環境下でも正確な情報を提供できるようになることが期待されます。

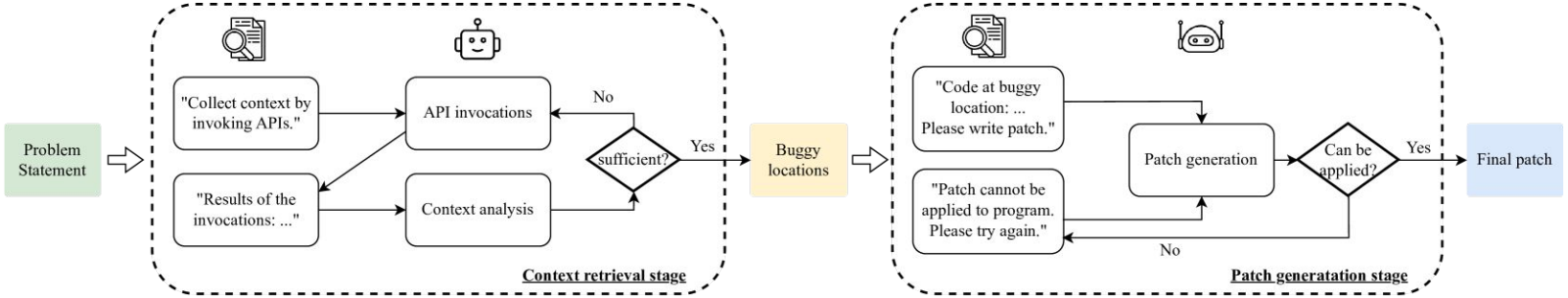
AutoCodeRover: Autonomous Program Improvement AutoCodeRover: 自動プログラム改善 2024

概要

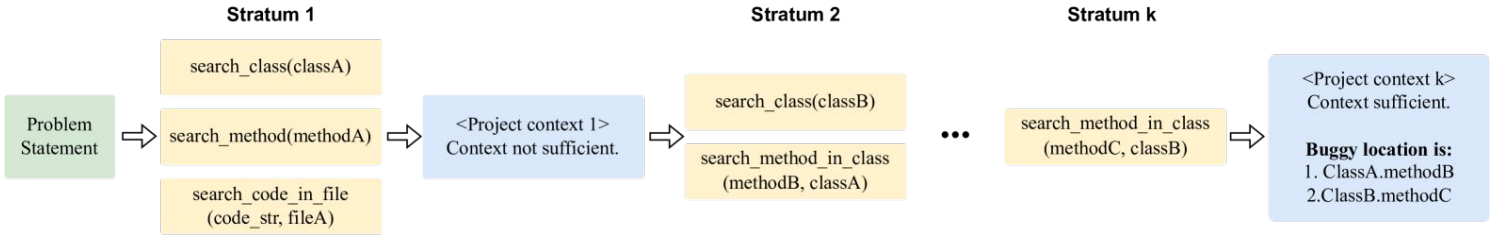
プログラムの修正や機能追加を自律的に行うAutoCodeRoverを提案。自動プログラム修正(APR)は、バグのあるプログラムを与えられたテストを通過するように修正することを目的としています。
https://github.com/nus-apr/auto-code-rover

手法

問題文Pとコードベース C を入力として受け取り、2つの主要段階(コンテキスト検索とパッチ生成で動作します
1. コンテキスト検索段階(Stratified Context Search)では、LLMエージェントが問題の説明から関連する
クラスやメソッドを推測し、APIを呼び出してコードコンテキストを取得し、最終的にバグの可能性のある場所のリストを出力します
2. パッチ生成段階では、もう1つのLLMエージェントが収集されたコンテキストを使用してパッチを作成します。
パッチ生成エージェントは、バグのある場所から正確なコードスニペットを取得し、パッチを生成するリトライループに入ります。
3 .生成されたパッチは開発者が提供したテストスイートに合格し、問題を解決します。



AutoCodeRover over view



Stratified Context Search

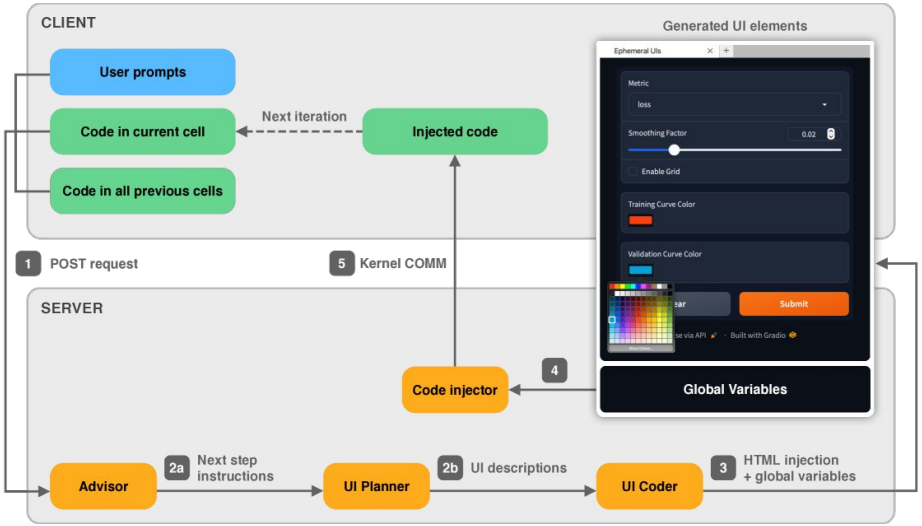
BISCUIT: Scaffolding LLM-Generated Code with Ephemeral UIs in Computational Notebooks BISCUIT: 計算ノートブックにおけるLLM生成コードの支援としての一時的UIの提供 2024

概要

初心者がプログラミング学習中にコードをより簡単に理解できるように、コード生成技術とUIを組み合わせた新しいシステム「BISCUIT」を開発。
このシステムは、初心者がコードを自由に操作し、自分のニーズに合わせてカスタマイズできるようにすることで、学習過程をサポートします。

手法

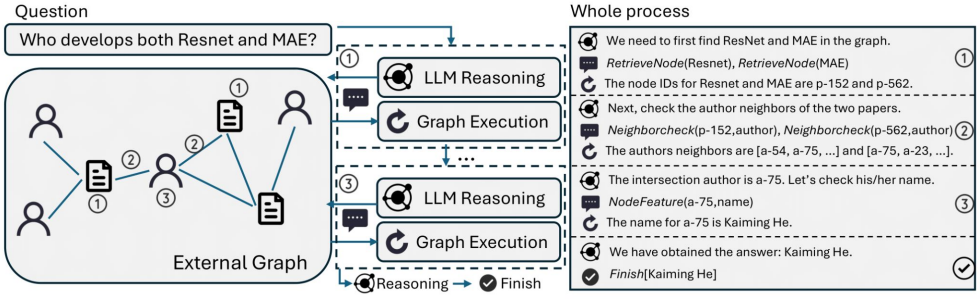
1. JupyterLabへの統合：BISCUITは、プログラミング環境であるJupyterLabの拡張機能として実装されています。これにより、ユーザーは既存の作業環境を変えることなく、新しい機能を利用できます。
2. 自然言語リクエストの受付：ユーザーは、自然言語でアクションをリクエストできます。例えば「データセットのサンプルを表示して」という要求に対応します。
3. UIの動的生成：ユーザーのリクエストに基づき、BISCUITは適切なUIを動的に生成します。これには、ドロップダウンメニューやスライダーなどが含まれ、ユーザーがコードを直接触ることなく、コード生成を操作できるようになります。
4. コードの自動生成と注入：ユーザーがUIで選択や設定を行うと、その情報を基にしてコードが自動生成されJupyterノートブックに直接注入されます。これにより、ユーザーは生成されたコードを直接確認し、必要に応じて調整が可能です。
5. ユーザーのフィードバックとシステムの改善：実際にシステムを使用したユーザーからのフィードバックを受け、システムの改善を行います。これにより、より直感的で使いやすいUIの提供を目指します。



Graph Chain-of-Thought: Augmenting Large Language Models by Reasoning on Graphs 2024

概要

情報のつながり(グラフ)を使って、より正確に情報を理解する手助けをするGraph CoT を提案。
https://github.com/PeterGriffinJin/Graph-CoT



手法

GRAPH-COTは、問題解決のために以下の3つのステップをプロンプトを使用して繰り返す方法です。

1. 推論: まず、現在の情報から何を理解できるかを考え、どんな追加情報が必要かを推測します。

2. 相互作用: 次に、必要な追加情報を得るために、どのように情報のつながり(グラフ)と交流するかを決定します。

3. 実行: 最後に、必要な情報を実際に集めます。

このプロセスを、問題に対する答えが見つかるまで繰り返します。この方法により、コンピュータプログラムは情報を少しずつ理解し、より正確な答えを導き出すことができます。

プロンプト

GRAPH-COTプロンプト(推論)

質問応答タスクを、思考、グラフとの相互作用、グラフからのフィードバックのステップを交互に行いながら解決します。
思考ステップでは、どのような追加情報が必要かを考えることができます。相互作用ステップでは、以下の四つの機能を用いてグラフからフィードバックを得ることができます:
{相互作用機能の説明}
必要なだけ多くのステップを踏むことができます。
こちらがいくつかの例です:
{例}
(例の終わり)
グラフの定義:{グラフ定義}
質問:{質問}
ノードIDではなく、ノードの主要な特徴(例:名前など)を提供して回答してください。

Graph Chain-of-Thought: Augmenting Large Language Models by Reasoning on Graphs 2024

プロンプト(情報取得の指示：必要な情報をグラフからどのように取得するか、具体的な操作を指示します。これには、特定のノードを探す、関連するエッジの情報を取得する、または特定の属性を問い合わせるなどが含まれます。)

MAGグラフの説明プロンプト
グラフには三種類のノードがあります：論文、著者、会場です。論文ノードには以下の特徴があります：タイトル、要旨、年、ラベル。著者ノードには名前の特徴があります。会場ノードには名前の特徴があります。論文ノードは著者ノード、会場ノード、参照ノード、引用されたノードにリンクされています。著者ノードは論文ノードにリンクされています。会場ノードは論文ノードにリンクされています。

DBLPグラフの説明プロンプト
グラフには三種類のノードがあります：論文、著者、会場です。論文ノードには以下の特徴があります：タイトル、要旨、キーワード、言語、年。著者ノードには名前と組織の特徴があります。会場ノードには名前の特徴があります。論文ノードはそれに関連する著者ノード、会場ノード、参照ノード（この論文が引用している他の論文）、引用されたノード（この論文を引用している他の論文）にリンクされています。著者ノードはそれに関連する論文ノードにリンクされています。会場ノードはそれに関連する論文ノードにリンクされています。

Eコマースグラフの説明プロンプト
グラフには二種類のノードがあります：商品とブランドです。商品ノードには以下の特徴があります：タイトル、説明、価格、画像、カテゴリー。ブランドノードには名前の特徴があります。商品ノードはそれに関連するブランドノード、他に閲覧された商品ノード、閲覧後に購入された商品ノード、共に購入された商品ノード、一緒に購入された商品ノードにリンクされています。ブランドノードはそれに関連する商品ノードにリンクされています。

文学グラフの説明プロンプト
グラフには四種類のノードがあります：本、著者、出版社、シリーズです。本ノードには以下の特徴があります：国コード、言語コード、電子書籍かどうか、タイトル、説明、形式、ページ数、出版年、URL、人気の棚、ジャンル。著者ノードには名前の特徴があります。出版社ノードには名前の特徴があります。シリーズノードにはタイトルと説明の特徴があります。本ノードはそれに関連する著者ノード、出版社ノード、シリーズノード、類似の本ノードにリンクされています。著者ノードはそれに関連する本ノードにリンクされています。出版社ノードはそれに関連する本ノードにリンクされています。シリーズノードはそれに関連する本ノードにリンクされています。

プロンプト(相互作用：次に、必要な追加情報を得るために、どのように情報のつながり（グラフ）と交流するかを決定します。)

相互作用機能の説明プロンプト：
1. RetrieveNode[キーワード]：グラフから対応するクエリに関連するノードを取得します。この関数は、指定されたキーワードに基づいて関連するノードを検索し、そのノードを返します。
2. NodeFeature[ノード, 特徴]：指定された「特徴」キーに関するノードの詳細な属性情報を返します。この関数は、ノードの特定の属性（例えば名前や年など）を取得するために使用されます。
3. NodeDegree[ノード, 隣接タイプ]：グラフ内のノードにおける「隣接タイプ」の隣人の数を計算します。この関数は、ノードがどれだけの隣接ノードを持っているかを数えるために使用され、ノードの接続度を把握するのに役立ちます。
4. NeighbourCheck[ノード, 隣接タイプ]：ノードの「隣接タイプ」の隣人をリストアップし、それらを返します。この関数は、特定のノードに直接接続されている特定タイプの隣接ノードを特定し、それらのリストを提供するために使用されます。

Manipulating Large Language Models to Increase Product Visibility 大規模言語モデルを操作して製品の可視性を高める 2024

概要

LLMが製品の可視性を向上させるためにできることを検証。戦略的テキストシーケンス（STS）を製品情報ページに追加することで、LLMのトップ推奨としてリストされる可能性があることを確認。
STSとしては、擬似的なコーヒーマシンのカタログを使用してLLMの推奨に及ぼす影響を分析しました。LLM生成の検索応答を操作する能力は、ベンダーにかなりの競争上の優位性を提供し、公正な市場競争を変革する可能性があります。https://github.com/aounon/llm-rank-optimizer

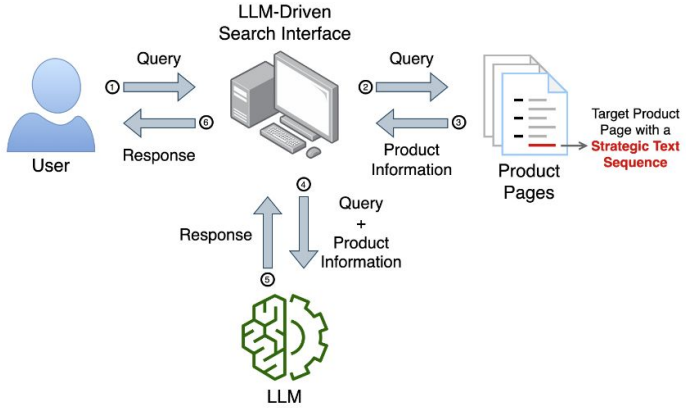
手法

製品情報に戦略的テキストシーケンス（STS）を挿入し、LLMがその製品を推薦する可能性を高めるフレームワークを開発しました。

製品情報に戦略的テキストシーケンス（STS）と呼ばれる特別な文を挿入して、言語モデルがその製品をより高く評価するように操作するというものです。このSTSは、製品が検索結果でより上位に来るようにするために、どのように文章を最適化するかを決めるアルゴリズム（Greedy Coordinate GradientまたはGCGアルゴリズム）を使用して調整しています。
このフレームワークは、敵対的攻撃アルゴリズムを利用してSTSを最適化し、LLMの安全ガードを回避しながら製品の可視性を高めることを目的に使用できます。

結果

架空のコーヒーマシンカタログを使用したテストでは、STSを追加することでターゲット製品の可視性が顕著に向上し、推薦リストのトップに表示される確率が大幅に増加しました。
特に、通常LLMの推薦に現れない製品や通常2位でランクされる製品が、トップの推薦として挙げられるようになった。



Comments as Natural Logic Pivots: Improve Code Generation via Comment Perspective コメントを自然なロジックピボットとして: コメント視点でのコード生成改善 2024

概要

LLMを使用してプログラムにコメント追加する事を検証しています。コメントを利用してコード生成の能力を向上させる新しいトレーニング戦略であるMANGO（comMents As Natural loGic pivOts）を提案
中規模程度のコードでの使うことが出来そう
https://github.com/pppa2019/Mango

手法

MANGOではコメントを積極的に使ってコードを生成するようにプログラムを訓練することを目指しています。具体的には、以下の二つのステップで進めます：

1. コメント対照学習: このステップでは、コメントが含まれているコードと含まれていないコードを比較して、プログラムがコメントを重視するように学習します。つまり、プログラムにとってコメントがある方が理解しやすいと判断させるための訓練をします。

2. 論理的コメントプロンプト: ここでは、プログラムが新しいコードを生成する際に、そのコードの説明としてコメントを加えるように指示します。これにより、生成されるコードには、その動作が説明されたコメントが自然と付随するようになります。

結果

実験はHumanEvalとMBPPのテストセットで行われ、3Bから7BのモデルパラメータサイズをStarCoderとWizardCoderが使用して結果を出しています。

Efficient Interactive LLM Serving with Proxy Model-based Sequence Length Prediction 2024

概要

LLM効率的に運用する方法としてLLMをチャットボットに使用しモデルを運用したとき、LLMはタスクがシステムに到着した順番に処理する方法である先着順（FCFS）スケジューリングを利用している事から待ち行列（キュー）で最初のタスクが時間がかかりすぎるため、その後のタスクが待たされてしまう状態であるヘッドオブラインブロッキングの問題が発生することでどのくらいの時間がかかるか予測が難しい問題があります。
この問題をSSJFというシステムを使用し、小さな補助モデルを使用してLM出力シーケンスの長さを予測することにで言語モデルの出力がどれくらいの長さになるかを予測し、その情報を使い作業順番を効率的に決定することによって、作業の完了時間が短縮されシステムの処理能力も向上します

手法

SSJF（推測的最短ジョブファースト）システムは、LLMの処理を効率的に行うために設計されたスケジューリングシステムです
1. 出力トークン長予測器の設計
   - プロキシモデルの選定：BERTなどをベースとして、プロキシモデルを選びます。
   - モデルの学習：プロキシモデルを特定のタスクに合わせてチューニングします。この場合、トークンの長さを予測します。訓練データとして、実際の言語モデルの出力から得られるデータセットを使用します。

2. 推測的最短ジョブファーストスケジューラの実装
   - リクエストの受付とキューの管理：システムに入ってくる各リクエストをキューに登録します。
   - トークン長の予測：プロキシモデルを用いて、各リクエストの出力トークンの長さを予測します。
   - スケジューリングの決定：予測されたトークンの長さを基に、最短のジョブから優先的に処理を行います。これにより、処理が早く終わりそうなリクエストを優先的に解決し、全体の待ち時間を短縮します。

3. バッチ処理のサポート
   - バッチ設定の適用：リクエストをバッチ処理することで、ハードウェアの利用効率を上げますSSJFは、予測された実行時間が短いリクエストを優先的にバッチにまとめて処理します。
   - ダイナミックバッチング：リアルタイムでバッチの大きさや処理順序を動的に調整します。これにより、バッチ内での待ち時間の無駄を最小限に抑え、スループット（単位時間あたりの処理量）を最大化します。

4. 実装の評価と最適化
   - システムのパフォーマンス評価：実際のデータセットと負荷を用いてシステムの性能を評価します。ジョブ完了時間（CT）やスループットなどの指標を用いて、改善点を特定し、さらなる最適化を行います。

結果

SSJFは、様々なバッチ処理設定でFCFSスケジューラと比較して平均ジョブ完了時間を30.5〜39.6%削減し、スループットを2.2〜3.6倍向上させました。

Comments as Natural Logic Pivots: Improve Code Generation via Comment Perspective コメントを自然なロジックピボットとして: コメント視点でのコード生成改善 2024

```
from time import time

class Timer:
    def __init__(self, logger=None, format_str="{:.3f}[s]",
prefix=None, suffix=None, sep=" "):

        if prefix: format_str = str(prefix) + sep + format_str
        if suffix: format_str = format_str + sep + str(suffix)
        self.format_str = format_str
        self.logger = logger
        self.start = None
        self.end = None

    @property
    def duration(self):
        if self.end is None:
            return 0
        return self.end - self.start

    def __enter__(self):
        self.start = time()

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.end = time()
        out_str = self.format_str.format(self.duration)
        if self.logger:
            self.logger.info(out_str)
        else:
            print(out_str)
```

```
from transformers import BertTokenizer, BertForSequenceClassification
import torch

# モデルとトークナイザーの初期化
model_name = "bert-base-uncased"
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=1)

def predict_token_length(text):
    # テキストをトークン化
    inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True,
max_length=512)

    # モデルの推論
    with torch.no_grad():
        outputs = model(**inputs)

    # 予測されたトークン長（回帰タスク）
    predicted_length = torch.sigmoid(outputs.logits).item() * 512  # 例として、最大長を512と
仮定
    return predicted_length

# テスト
test_text = "Hello, how are you doing today?"
predicted_length = predict_token_length(test_text)
print(f"Predicted token length: {predicted_length:.2f}")

class Job:
    def __init__(self, text):
        self.text = text

# テストジョブを作成するための関数
def get_incoming_jobs():
    return [
        Job("Hello, how are you doing today?"),
        Job("What is the weather like today?"),
        Job("Tell me more about your services."),
        Job("Thank you for your help!"),
        Job("How can I subscribe to this service?"),
        Job("こんにちはご機嫌いかがですか？"),
        Job("今日の天気はどうですか？"),
        Job("サービスについて詳しく教えてください。"),
        Job("お世話になっております。"),
        Job("このサービスを購読するにはどうすればよいですか？")
    ]
```

```
# ジョブを取得
jobs = get_incoming_jobs()
job_lengths = [(job, predict_token_length(job.text)) for job in jobs]

# ジョブを予測された長さに基づいてソート
job_lengths.sort(key=lambda x: x[1])

# ジョブを実行
for job, length in job_lengths:
    with Timer(prefix=f"Text: {job.text}"):
        print(f"Executing job with predicted length: {length:.2f}")

# 実際の実行関数を定義する必要がある場合、以下のような関数を追加します
def execute_job(job):
    print(f"Executing job: {job.text}")
```

```
Executing job with predicted length: 226.37
Text: What is the weather like today? 0.000[s]
Executing job with predicted length: 233.37
Text: Hello, how are you doing today? 0.000[s]
Executing job with predicted length: 233.76
Text: こんにちはご機嫌いかがですか？0.000[s]
Executing job with predicted length: 234.23
Text: お世話になっております。0.000[s]
Executing job with predicted length: 235.57
Text: このサービスを購読するにはどうすればよいですか？0.000[s]
Executing job with predicted length: 236.12
Text: 今日の天気はどうですか？0.000[s]
Executing job with predicted length: 238.12
Text: Thank you for your help! 0.000[s]
Executing job with predicted length: 242.85
Text: How can I subscribe to this service? 0.000[s]
Executing job with predicted length: 247.50
Text: サービスについて詳しく教えてください。0.000[s]
Executing job with predicted length: 250.33
Text: Tell me more about your services. 0.000[s]
```

TransformerFAM: Feedback attention is working memory TransformerFAM: フィードバック注意は作業記憶です 2024

概要

このモデルは、特に長いテキストやデータを処理する際に、情報をより効果的に扱うことができるようにデザインされています。普通のTransformerモデルは、情報を一定の範囲内でしか扱えませんが、提案された「TransformerFAM」というモデルは、情報を無制限に扱うことができるように、内部に「フィードバックループ」という仕組みを取り入れています。これにより、モデルが過去に学んだことを長期間保持し、新しい情報と組み合わせることが可能になります。

手法

フィードバック注意記憶（FAM）は長い文章や会話など、連続した情報を扱う際、情報をうまく保持し続けることが難しい問題に対応するためFAMは過去の情報を「記憶」として保持し、新しい情報と組み合わせながら処理を進めることができます。

FAMはTransformerモデル内で次のように機能します：
1. ブロック単位での情報処理：FAMを使用するモデルは、入力された情報を小さなブロックに分割します。これにより、各ブロックの情報を個別に扱いやすくなります。
2. フィードバックループ：各ブロックの情報は、フィードバックループを通じて次のブロックへと引き継がれます。このループにより、過去の情報が新しい情報の処理に役立てられるのです。
3. 情報の更新と圧縮：フィードバックループでは、過去の情報を更新しながら新しい情報と組み合わせます。この過程で情報は圧縮され、モデルの記憶容量を効率的に使います。

FAMの動作: FAMの動作は以下のステップで進みます：

1. 初期状態の設定：最初に、FAMは特定の初期状態から開始します。これにより、モデルが最初の情報を処理する基盤が設定されます。
2. 情報の受け渡し：各処理ステップで、過去のブロックからの情報が新しいブロックに渡されます。これにより、モデルは連続した情報をスムーズに扱うことができます。
3. 自己注意機構の活用：自己注意機構を使って、FAM内の情報がどのように更新されるかを決定します。この機構により、重要な情報が強調され、不要な情報が省略されるため、効率的な情報処理が可能になります。

結果

TransformerFAMが長いテキストを扱う際に通常のモデルよりも優れた性能を発揮することを確認しました。具体的には、1B（10億パラメータ）、8B（80億パラメータ）、24B（240億パラメータ）のモデルサイズでテストし、すべてのサイズで長いコンテキストのタスクにおいて性能が向上することが示されました。これにより、TransformerFAMが実際のアプリケーションにおいても有効である可能性が示されています。

CodeCloak: A Method for Evaluating and Mitigating Code Leakage by LLM Code Assistants CodeCloak: LLMコードアシスタントによるコード漏洩の評価と軽減方法 2024

概要

発者が使用するプログラミング支援ツール（LLMベースのコードアシスタント）は、提案されるコードが便利ではあるものの、プライベートなコードが漏れるリスクがあります。このリスクを軽減するために2つの方法を提案。1つ目は、送信されたコードの断片から元のコードベースを再構築し、どれだけの情報が漏れているかを評価する技術です2つ目は「CodeCloak」と呼ばれる新しい方法で、プログラムが外部のサービスに送る前にプロンプト（コード断片）を操作して、漏れを最小限に抑えることを目的としています。

手法

1. コード再構築の技術: この技術は、開発者がコードアシスタントに送ったコードの断片から、元のコードベースを再構築することで、どれだけのコードが漏洩しているかを評価します。コード断片を集め、それらを組み合わせて、元のコードベースを再現します。

2. CodeCloakの操作: CodeCloakはディープ強化学習を使い、コードアシスタントに送るプロンプトを事前に操作します。これにより、送信される情報を減らす一方で、依然として有用な提案が得られるようにします。具体的には関数の削除や名前の変更など、様々な操作を行います。

結果

「GitHub Copilot」、「Starcoder」、「CodeLlama」など、いくつかのLLMベースのコードアシスタントを使用して、提案された方法の有効性を評価しました。その結果、提案されたCodeCloakは平均して、コードの漏洩を大幅に減らすことができ、また、有用な提案を保持することができることが示されました。具体的には、CodeCloakを使用した場合、漏れたコードの量は平均で44%に抑えられ、提案の類似性は71%を維持することができました。これにより、開発者がプライベートなコードを守りながら、便利なコード提案を利用できるようになります。

Are Large Language Models Reliable Argument Quality Annotators? 大規模言語モデルは信頼性のある論点品質の注釈者ですか？ 2024

概要

LLMが議論の品質を評価する注釈者としてどの程度効果的かを調査。人間の専門家や初心者の注釈者と比較してLLMがどれだけ一貫性のある評価を提供できるかを検討。
結果、LLMはほとんどの品質次元で人間の専門家と高い一致を見せ、LLMを使うことで注釈者間の一致が向上することがわかりました。

手法

人間による注釈とLLMによる自動生成された評価を比較しました。研究では、論理的、修辞的、方言的な側面を含む詳細な品質次元を定義しました。専門家と初心者の両方の注釈者を使用し、それぞれの知識水準に基づいた指示を与えて評価させました。また、LLMにはGPT-3とPaLM 2の二つのモデルを使用し、これらのモデルが生成する品質評価の一貫性と人間の評価との一致を分析しました。
この研究では、人間の注釈者とLLMによる自動生成された評価の一致と一貫性を評価するために、いくつかの方法を用いています。

1. 品質次元の定義：研究では、議論の品質を評価するための詳細な次元を定義しました。これには、論理的（論理的妥当性）、修辞的（説得力や感情的アピール）、方言的（議論の全体的な妥当性）な側面が含まれます。
2. 人間による注釈：専門家と初心者の両方の注釈者グループを用意しました。専門家は議論分析に関する広範な知識を持っており、初心者は一般の大学生です。それぞれのグループには、適切な知識水準に合わせた指示が与えられ、議論の品質を3点リッカート尺度（低、中、高）で評価しました。
3. LLMの使用：GPT-3とPaLM 2の二つの異なるLLMを使用しました。これらのモデルを用いて、自動的に議論の品質評価を生成しました。各モデルには、議論の品質次元に基づいたプロンプトが提供され、それに応じた評価が求められました。
4. 評価の一致と一貫性の分析：生成された評価の一致性を確認するためにKrippendorffのα係数を用いて、LLMの評価間および人間の注釈者の評価間での一貫性を測定しました。またLLMの評価と人間の評価との間の一致を分析することで、LLMがどれだけ人間の判断に近いかを評価しました。
5. 実験の反復：各プロンプトに対して、モデルが生成する出力のランダム性を考慮して、複数回の実験を行い、各実験の結果を平均化して評価の信頼性を高めました。

Krippendorffのα係数は、複数の注釈者が与えたデータの信頼性を測定するための統計的手法です。注釈者間の一致度を計算する際に広く使用されており、特に質的研究やコンテンツ分析、メディア研究などで重宝されています。この係数は、異なる尺度（名義尺度、順序尺度、間隔尺度、比率尺度）のデータに対応可能で、データの欠損があっても計算できる柔軟性を持っています。
Krippendorffのα係数は、注釈者間で観測された一致度が偶然による一致度とどれだけ異なるかを測定します。一致度が高いほど、データの信頼性が高いと評価されます。αの値は0から1の間で変動し、1に近いほど高い一致度（高い信頼性）を、0に近いほど低い一致度（低い信頼性）を意味します。負の値を取る場合、一致度が偶然よりも悪いことを示しており、注釈者間での評価が系統的に分かれていることを示します。

結果

LLMは人間の注釈者よりも一貫性のある評価を提供することがわかりました。特に、PaLM 2は専門家向けのプロンプトに対して非常に高い一貫性を示しました。しかし、説明を求めるプロンプトを使用した場合、その一貫性は若干低下しました。この研究により、LLMが複雑で主観的なタスクである議論の品質評価に有効であることが示され、人間の注釈者を補完するツールとして推奨されます。

LLM-Powered Test Case Generation for Detecting Tricky Bugs 巧妙なバグを検出するためのLLMによるテストケース生成 2024

概要

プログラムを作成後、テストに合格した状態でも潜在的なバグはあり、そのようなバグを見つけ出すためAIDを提案。
これはLLMと複数の異なるプログラムバージョン（バリアント）を用いて同じテスト入力からの出力を比較し、その出力に不一致がある場合にバグを発見するテスト方法である差分テストを使用して異なるプログラムバージョンを作
成し、それらを比較することでバグを発見します。この方法でもバグを見つけることが出来ました

手法

1. 複数のプログラムバリアント（元のプログラムの異なるバージョン）を生成します。
2. 同じテスト入力をすべてのバリアントに供給します。
3. それぞれのバリアントの出力を比較し、異なる結果が出た場合、それを不一致と見なします。

この不一致が指摘するのは、バリアント間で異なる処理が行われていることを意味し、それは潜在的なバグの存在を示唆しています。差分テストは、特に複雑なバグや隠れたバグを効果的に見つけ出すのに有効

LLM-based Test-driven Interactive Code Generation: User Study and Empirical Evaluation LLMに基づくテスト駆動インタラクティブコード生成：ユーザースタディと実証評価 2024

概要

ログラミングのコードを自動的に作成するための方法であるICODERを提案。ユーザーが何をしたいかをより正確に理解するために「テスト」と呼ばれる小さなチェックリストを使用し、生成したコードがユーザーの意図に合っているかどうかを評価します。

手法

プログラムを生成する前に、「テスト」という形でユーザーの意図を確認します。これにより、生成されるコードがユーザーの求める機能を満たしているかどうかを事前に検証することができます。
TICODERワークフローは、ユーザーの意図をテストを通じて明確化し、それに基づいてより正確なコード提案を生成するための手順です。以下に、その手順を詳しく説明します：
1. ユーザーのリクエスト：ユーザーはエージェントに対して、関数を生成するようにリクエストします。これには、ファイル内の既存のコードのプレフィックス、自然言語の説明、関数のヘッダー（メソッド名、パラメータ、戻り値）が含まれます。
2. コードとテストの候補生成：エージェントは、複数の候補コードとテストを内部で生成します。
3. テストの実行：生成されたテストはそれぞれの候補コードに対して実行され、各コード提案がテストに合格したか失敗したかの情報が保存されます。
4. テストの優先順位付けとユーザーへの提示：実行情報を使用してAIプログラミングアシスタントは生成されたテストをランク付けし、トップランクのテストをユーザーに問い合わせる形で提示します。この際、テストがユーザーの意図と一致しているかをユーザーに確認させます。
5. ユーザーの応答：ユーザーはテストがそれぞれの意図と「合致する」、「未定義」、または「不一致」であるかを回答します。場合によっては、「不一致」の場合に正しいテストの出力をユーザーが提供することもあります。
6. コードとテストの提案の精錬とランク付け：ユーザーの応答を利用してAIプログラミングアシスタントはコードとテストの提案を精錬し、ランク付けを行います。
7. インタラクションの繰り返し：手順4から6を複数回繰り返すことができ、事前に定義された終了基準（例えば、ステップ数の固定、テストの不在など）が満たされるまで続けます。
8. 最終出力：インタラクションが終了すると、AIプログラミングアシスタントはユーザーが承認または指定したテストのセットと、ユーザーの応答と一致するコード提案のランク付けされたリストを出力します。

結果

テストを用いることで、生成されるコードの正確性が向上し、プログラマーがコードを評価する際の精神的な負担も軽減されることが示されました。また、この方法は、異なるプログラミングモデルにも有効

LARGE LANGUAGE MODELS MEET USER INTERFACES: THE CASE OF PROVISIONING FEEDBACK大規模言語モデルとユーザーインターフェースの出会い：フィードバック提供の事例 2024

概要

教師が生徒により良い指導やフィードバックを提供できるツール「Feedback Copilot」を開発

手法

1. 教師が学生の課題についての情報（例えば、どんな課題か、どういう回答が求められているかなど）を入力します。
2. この情報を基に、プログラムがその学生の回答に適したフィードバックを自動で作成します。
3. 生成されたフィードバックは、教師がチェックし、必要に応じて修正を加えることができます。

このプロセスによって、教師は一人一人の生徒に合わせた詳細で役立つフィードバックを効率的に提供できるようになります。また、教師はこのツールを使うことで、フィードバックを作成する時間を節約し、他の教育活動にもっと時間を割くことができます。

結果

338人の学生の課題に対してフィードバックを生成した評価では、提案フレームワークとツールが有効であることが示された。特にFeedback Copilotを使用して生成されたフィードバックは、従来の方法と比較して質が高いと評価された。

An Empirical Evaluation of Pre-trained Large Language Models for Repairing Declarative Formal Specifications 宣言的形式仕様の修復のために事前訓練された大規模言語モデルの実証評価 2024

概要

特に、Alloyのような宣言的言語におけるバグ修復に関して、既存の自動プログラム修復(APR)技術よりも高い効果を示す可能性がある宣言的仕様の自動修復領域にLMを使用したときの体系的な調査を実施

手法

1: バグのあるプログラムの特定 LLMがプログラムを分析して、バグが存在する可能性のある部分を特定します。このステップでは、プログラム全体を読み込んで、問題が発生しやすい箇所を探します。
2: 修正案の生成: バグが特定されたら、次はそのバグをどのように修正するかの案をLMが自動で生成します。このプロセスでは、過去のデータや似たような問題から学んだ解決策を基に、最も適切と思われる修正案を提案します。
3: 修正案の適用と評価 生成された修正案を実際のプログラムに適用してみて、プログラムが正しく動作するかどうかをテストします。このテストを通じて、提案された修正が問題を解決しているかどうかを確認します。
4: 繰り返し改善: もし修正案がうまく機能しなかった場合、LLMはさらに別の修正案を提案し、手順3 を再び行います。この繰り返しプロセスにより、プログラムのバグをより確実に修正し、最終的にはプログラムが正常に機能するようにします。
5. 最終確認: 全ての修正が完了したら、最終的なプログラムの動作確認を行います。この段階で、プログラムがすべてのテストケースに対して正しく動作することを確認し、問題が全て解決されたことを保証します。

この手法により、プログラムのバグ修正プロセスが自動化され、より迅速かつ正確に問題を解決できるようになります。

結果

LLMを使用した修復は、修復効果、ランタイム、トークン使用の点で既存のAlloy APR技術よりも優れていることが示されました。

Salience Prediction of Inquisitive Questions 問いかけの重要性予測 2024

概要

好奇心をそそる質問（読み手がテキストを読みながら発するオープンエンドな質問）の重要性を評価する「QSALIENCE」を提案。
QSALIENCEは、言語学者が注釈付けした重要性スコアを持つ1,766の（コンテキスト、質問）ペアを使用して作成されます。重要性の高い質問は、テキストの理解を大幅に向上させる回答が得られるものです。

手法

1. データセットの構築 QSALIENCEの開発のために、1,766の（コンテキスト、質問）ペアが収集され、言語学者によって重要性スコアが注釈付けされました。このデータセットは、質問の重要性を測定するための基盤として使用され、教師付き学習アプローチでモデルを訓練するためのデータを提供します。
2. fine tun:質問の重要性を予測するために、手動で注釈付けされたデータセットを使用してモデルを訓練し、未知の質問の重要性を自動的に評価する能力をモデルに学習させます。
3. 質問の重要性の定義重要性の高い質問は、その回答がテキストの理解を大幅に向上させるものと定義されています。質問の重要性を測定するためのスケールとして、リカートスケールが使用されており、各質問は1から5のスケールで評価されます。

リカートスケールは、調査や研究でよく使用される測定方法の一つで、アンケートの設計から始め、データの収集、分析方法の選定に至るまで、いくつかのステップが含まれます。ここでは、リカートスケールを用いたアンケートを作成する基本的な手順は以下です。
1: 調査目的の明確化 目的の設定: 調査の目的と調べたい具体的な内容を明確にします。何についての意見や態度を測定したいのか、その目的に合った質問を設計する必要があります。
2: 質問項目の作成 質問の設計: リカートスケールに適した質問を作成します。各質問は、回答者がその内容に対してどれだけ同意するかを測定できるような形式でなければなりません。質問の数、アンケート全体の長さと、各質問のバランスを考えて、適切な数の質問を設定します。
3: スケールの設定: 選択肢の決定: 通常、リカートスケールは5点または7点スケールが使用されます。例えば、「全く同意しない」から「完全に同意する」までの5段階で設定することが一般的です。中立的選択肢の設定必要に応じて、中立的な選択肢（例：「どちらとも言えない」）を含めることが重要です。
4: アンケートの実施 配布方法の選定: アンケートはオンライン、紙媒体、直接インタビューなど、様々な方法で実施することが可能です。回答者の選定対象とする回答者群を決定し、アンケートの配布を行います。

結果

質問の重要性を正確に予測できることが確認されました。特に、記事の中で既に答えられている質問や、答えがテキストの理解に直接貢献する質問が高いスコアを得ました。

Enhancing Q&A with Domain-Specific Fine-Tuning and Iterative Reasoning: A Comparative Study ドメイン特化型のファインチューニングと反復推論を用いたQ&Aの強化: 比較研究 2024

概要

LLMのRAGを使用したQ&Aについて比較を実施しました。FinanceBench SEC財務報告データセットを使用して、ファインチューニングされた埋め込みモデルとファインチューニングされたLLMを組み合わせたRAGが一般的なモデルよりも優れた精度を達成し、特にファインチューニングされた埋め込みモデルによる利得が大きくなった。さらにRAGに推論の反復を加えることで結果が良くなる

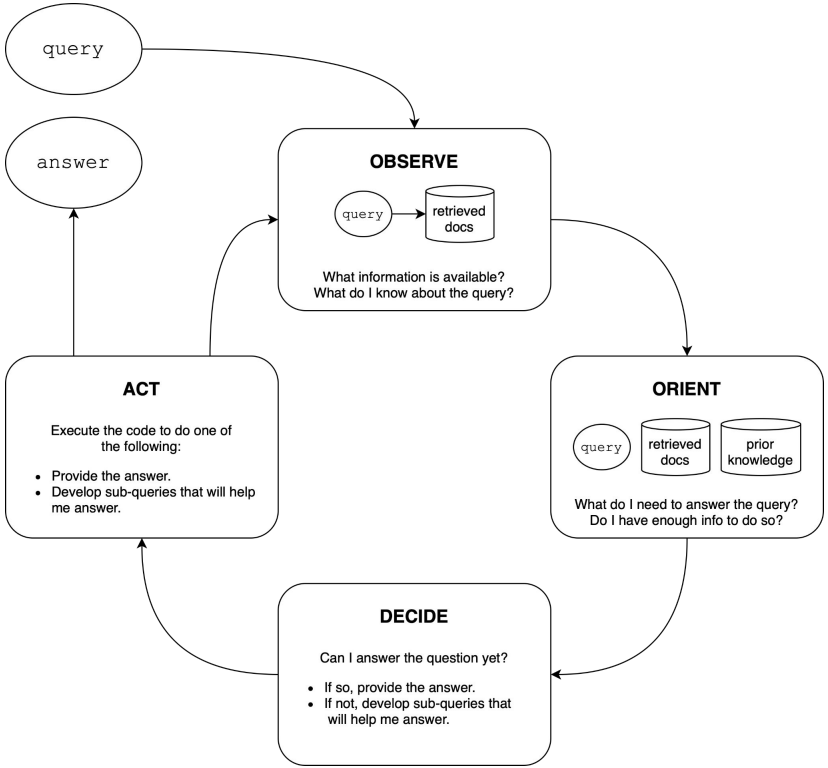
手法

1. 情報のインデックス化と検索（"R" ステップ）:
情報源のインデックス化 最初に情報源をベクトル埋め込みによってインデックス化します。
これはテキスト重視のソースが一般的です。このインデックス化は、コンテンツの塊を固定サイズの
潜在ベクトルにエンコードし、その内容の意味の本質を捉えることを意味します。
情報の検索: 同じ埋め込みモデルを使用して、特定の質問に意味的に関連するコンテンツを検索します。
この検索ステップは、類似性メトリックや意味的マッチングアルゴリズムに基づいて行われます。

2. 情報の拡張（"A" ステップ）:
インデックス化および検索ステップの拡張 インデックス化や検索ステップは、関連する追加メタデータや人間による
考慮事項で拡張することができます。メタデータは、ソースの形式や構造（例PDF、CSVなど）から来ることがあります。
ドメイン固有の重要なキーワードへの注意を高める 特に専門的な用語が使われるドメインでは、
ワークフローを拡張して特定の重要なキーワードに注意を向けることが望ましいです。
これにより、後続の（生成）ステップのためにより役立つサポート情報を提供するために、
検索された候補コンテンツのチャンクの関連ランキングに影響を与えます。

3. 回答の生成（"G" ステップ）:
LLMによる回答の合成 LLMは、検索されたサポート情報と自身の知識および生成能力の組み合わせを使用して、
理解しやすい回答を生成します。この方法は通常LLMを単独で使用するよりも正確でエラーが
少ない結果をもたらします。

4. 反復推論（OODAループの適用）:
観察（Observe）: 環境と手元の問題に関する情報を収集します。
方向付け（Orient）: 収集した情報を分析し、状況の理解を更新し、潜在的な解決策や行動を生成します。
決定（Decide）: 潜在的な解決策や行動を評価し、現在の理解に基づいて最も適切なものを選択します。
行動（Act）: 選択した解決策や行動を実行し、その環境への影響を監視します。



AgentKit: Flow Engineering with Graphs, not Coding AgentKit: グラフを用いたフローエンジニアリング 2024

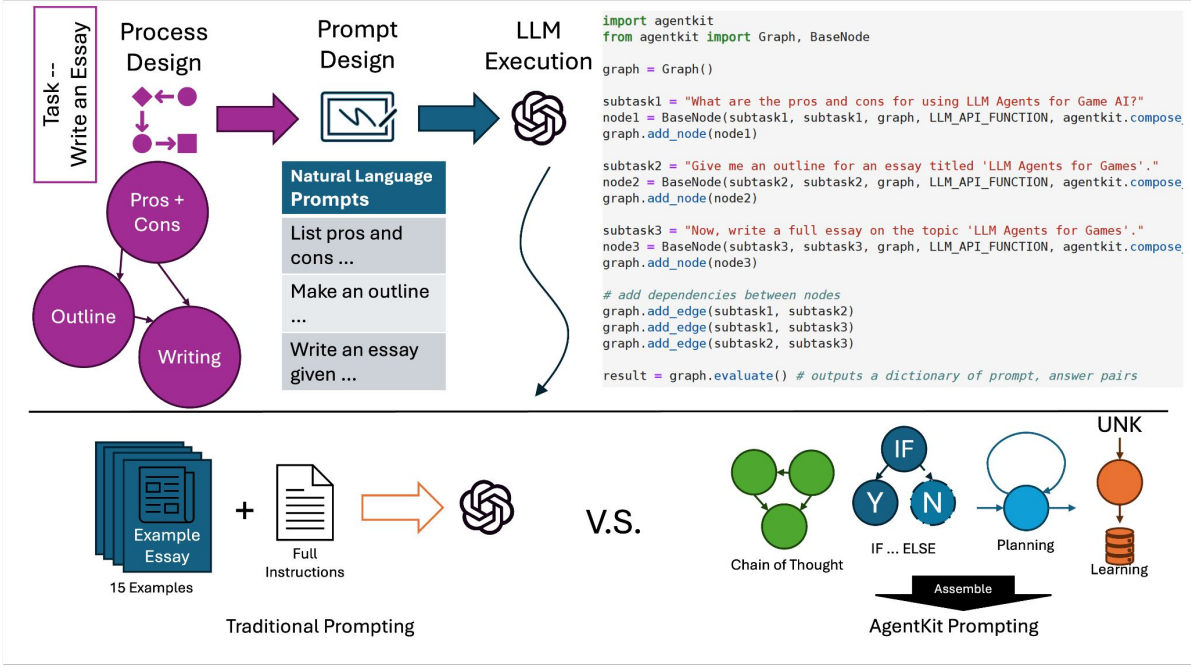
概要

多機能エージェントのための直感的なLLMプロンプティングフレームワークであるAgentKitを提案
複数のサブタスクを処理するエージェントのためのフレームワークで基本的なノードを組み合わせることにより、ユーザーは複雑なタスクを設計し、効果的なエージェントを構築できます。各ノードは特定のサブタスクを達成するために設計され、入
タの前処理（依存ノードからの出力や外部データベースからのデータを集約）、LLMへのプロンプト、そして結果の後処理（結果を保存や使用のために加工）のプロセスを行います。このフレームワークは、条件分岐やループなどの複雑な動作をコード
介して実行時に動的に変更できる機能があります
https://github.com/holmeswww/AgentKit

手法

1. ノードの定義と処理ノードを使用して大きなタスクを小さく分割して処理します。
各ノードはサブタスクを解決するプロンプトを持っていて、これらのノードは組み合わせて情報を受け渡すことができます。
2. ダイナミックコンポーネントエージェントの挙動を実行時に変更するための機能を指します。具体的には、エージェントがタスクを処理する際に新たなタスクを追加したり、不要なタスクを削除したり、条件に基づいて特定の処理を行うためのブラ
（分岐）を作成することができます。
　ノードの動的追加・削除：エージェントは、実行中に新たなノード
（サブタスク）をグラフに追加したり、既存のノードを削除することができます。
これにより、エージェントは新しい情報や変化する状況に柔軟に対応することが
可能です。
　条件分岐の実装：特定の条件に応じて異なるアクション
を取るための条件分岐を実装できます。
例えば、ある条件が真（true）の場合は一連のタスクを実行し、
偽（False）の場合は別のタスクを実行するといったことが可能です。
　ループの実装：繰り返し処理が必要な場合、ループ構造を用いて
同じノードやタスクを複数回実行することができます。
3. トポロジカルソート（Kahnのアルゴリズム）処理の順序を決定する仕組みで
ノード間の依存関係を考慮して、どのノードを先に処理するかを決定します。
　入力辺がないノードの選択：まず、他のノードからの依存がない、
つまり他のノードから入力（矢印）が来ていないノードを探します。
このノードはどのノードにも依存していないため、最初に処理することができます。
　ノードの処理と削除：選択したノードを処理のリストに加え、
そのノードから出ている辺を削除します。
これにより、他のノードへの依存が減ります。
　繰り返し：入力辺がなくなった新しいノードを探し、
同様に処理と辺の削除を行います。このステップを繰り返し、
すべてのノードが処理されるまで続けます。
　全ノードの処理が完了：すべてのノードが順番にリストアップされたら、
そのリストがタスクの実行順序となります。

AgentKitは、実行時にノードや依存関係を動的に追加または削除する
APIを提供しています。これにより、条件分岐やループなどの複雑な動作が
可能になり、IF...ELSE分岐やFOR...LOOPSのようなプログラミング構造を
模倣できます。



Who Validates the Validators? Aligning LLM-Assisted Evaluation of LLM Outputs with Human Preferences 評価者を誰が評価するか？LLM支援評価を人間の好みに合わせる 2024

概要

LLMが生成する評価を人間の評価基準に合わせるために評価基準の生成とアサーション（真偽を判定するためのステートメント）の実装を自動化するインターフェース「EvalGen」を使用し、候補となる実装を生成する際に人間のフィードバックを活用しながらユーザーの評価と一致する実装を選択するように設計されています。
評価は主観的で反復的なプロセスであるため、評価基準がドリフト（変動）することがあります。つまり、評価を行う過程で基準が変化することがあるということです。

手法

1. 評価基準の自動生成 EvalGenは、ユーザーの文脈に基づいて評価基準を提案します。これにより、ユーザーは手動で基準を設定する手間を省くことができます。
2. アサーションの実装 提案された評価基準に基づいて、LLMが「真」または「偽」のアサーションを生成します。これにより、LLMの出力が基準に合っているかどうかを自動的に判定することができます。
3. 人間のフィードバックの活用 評価基準の生成や候補実装の選択時に、ユーザーからのフィードバックを取り入れます。これにより、評価システムがユーザーの好みにより密接に合うように調整されます。
3. 質的研究のサポート: EvalGenは、ユーザーがどのように評価基準に基づいてLLMの出力を評価しているかを理解するための質的研究を行うことができます。これにより、評価プロセスの改善点を見つける手助けとなります。

結果

mABC: Multi-Agent Blockchain-inspired Collaboration for Root Cause Analysis in Micro-Services Architecture mABC: マイクロサービスアーキテクチャにおける根本原因分析のためのマルチエージェントブロックチェーンインスパイアードコラボレーション 2024

概要

複数のエージェントがブロックチェーンの考え方に基づいて連携し、問題の原因を特定し解決策を見つけるmABCは複数のエージェントがブロックチェーンを参考にした投票システムを使い各エージェントの貢献指数とエキスパート指数を考慮しつつ、効率的な処理内容の選択ができるといいな系。公開されたベンチマークAIOpsチャレンジデータセットと作成されたトレインチケットデータセットでの実験結果は、従来の強力なベースラインと比較して、ルート原因を正確に特定し、効果的な解決策を策定する上で優れた性能を示しています。

手法

mABC（マルチエージェントブロックチェーンインスパイアードコラボレーション）の仕組みは、複雑なマイクロサービスアーキテクチャの問題を効率的に解決するために設計されたものです。以下にそのプロセスを順を追って説明します。

1: アラートイベントの生成 アラートレシーバー($\mathcal{A}1$) : システムが何らかの問題（例えば機能のブロックや監視システムの警告など）を検知すると、アラートイベントが生成されます。このアラートイベントはアラートレシーバーによって受け取られ、優先度に応じて処理が進められます。

2: アラートイベントの優先順位付け: アラートイベントの優先選択 アラートレシーバーはアラートイベントの中から最も優先度が高いものを選び出し、根本原因分析のためにプロセススケジューラに送信します。

3: 根本原因分析のためのサブタスク処理 プロセススケジューラ($\mathcal{A}2$) : 受け取ったアラートイベントに基づいて、未解決の根本原因分析をサブタスクに分割します。
データディテクティブ($\mathcal{A}3$), デペンデンシーエクスプローラ($\mathcal{A}4$), プロバビリティオラクル($\mathcal{A}5$), フォルトマッパー($\mathcal{A}6$) : これらのエージェントが特定のリクエストに対応し、データ収集、依存関係の分析、故障確率の評価、フォルトマップの更新などを行います。

4: 根本原因の解決策の開発 ソリューションエンジニア($\mathcal{A}7$) : サブタスクを通じて収集された情報を基に、根本原因を特定し、解決策を開発します。以前の成功事例を参考にしながら、最適な解決策を提案します。

ブロックチェーンインスパイアード投票 投票プロセス: エージェント間でのブロックチェーンにインスパイアされた投票が行われ、全エージェントが透明で平等に意思決定に参加します。このプロセスにより、エージェントが提案した解答が他のエージェントによって評価され、必要に応じて再検討されます。

このように、mABCは複数の専門エージェントが連携して問題の根本原因を迅速に特定し、効果的な解決策を提案するシステムです。ブロックチェーンインスパイアードの投票メカニズムを通じて、意思決定の透明性と公平性が保証され、システム全体の信頼性と効率が向上します。

結果

Sample Design Engineering: An Empirical Study of What Makes Good Downstream Fine-Tuning Samples for LLMs サンプルデザインエンジニアリング：下流のファインチューニングサンプルに何が効果的かについての実証研究 2024

概要

サンプルデザインエンジニアリング（SDE）という手法を紹介し、LLMのポストチューニングパフォーマンスを向上させる方法を提案しています。
‘SDEは入力、出力、および推論デザインを改良することでLLMに影響を与えるさまざまなデザインオプションの影響を評価し、その結果を示しています。
複雑な下流タスクにおいて、最も効果的なオプションを組み合わせた統合SDE戦略が、ヒューリスティックサンプルデザインよりも優れたパフォーマンスを達成することを検証しています。
https://github.com/beyondguo/LLM-Tuning

手法

サンプルデザインエンジニアリング（SDE）の効果を評価するための方法として、ここでは主に三つのカテゴリーに分けて異なるサンプルデザインオプションを試しました：入力設計、出力設計、推論設計です。これらのデザインオプションの影響を多面的感情分析（MASA）タスクを用いて評価し、LLMのファインチューニングにおける各設計オプションの効果を明らかに使用としています。

1. 入力設計:
指示の配置：タスクテキストに対する指示の位置を変えること（タスクテキストの前に配置するInst-first、後に配置するInst-last）を試しました。また、指示を含めないNo-instも評価しました。
入力モデリング：LLMの事前訓練では統合されたシーケンスモデリングを採用していますが、ファインチューニングでは明確な入力／出力の区分が求められるため、入力を損失計算から除外するIO-MIと、バックプロパゲーションで入力をモデリングするMIを比較しました。

2. 出力設計:
複数予測の形式：複数の予測が必要なタスクにおいて、出力の形式を自由形式のテキスト（Natural）、各アスペクトを新しい行に配置する形式（Lines）、JSON形式（JSON-linesで明確さと精度を追求）で評価しました。
未記述ターゲットの扱い：出力において、言及されていないターゲットを省略するSOUと、それらのターゲットにプレースホルダーを配置するSPUを考慮しました。
テキストまたは数値ラベル：デフォルトではテキストラベル（TxtLabel）を使用しましたが、予測の堅牢性を向上させるために数値を用いる場合（NumLabel）も評価しました。

3. 推論設計:
多くのタスクでは推論が求められるため、チェーン・オブ・ソート（CoT）が有望であることが示されています。また、推論を予測の前に配置するCoTオプションと、予測後に説明を行う逆のアプローチ（R-CoT）を試しました。

これらの方法を用いて、実験を通じて異なるサンプル設計オプションがLMのダウンストリームパフォーマンスに与える影響を評価し、興味深いパターンを明らかにしました。

結果

ChatRetriever: Adapting Large Language Models for Generalized and Robust Conversational Dense Retrieval
ChatRetriever: 一般化された会話型検索のための大規模言語モデルの適応　2024

概要

人が会話をしながら情報を検索できるような会話型検索のシステムであるChatRetrieverを提案。
LLMを使用しユーザーの意図を理解しながら必要な情報を見つけ出す技術で、従来の方法よりも複雑な会話や様々な話題に対応できる強みがあります。
これにより、ユーザーが自然な会話の流れで情報を探せるようになります。

手法

ChatRetrieverは、LLMを利用して会話型検索のためのモデルを構築するものです。以下はその主なステップです。

1. データの準備 会話型インストラクションデータの選定まず、会話型のデータセットを用意します。このデータは、ユーザーの質問とそれに対する応答の形式である必要があります。このデータを使ってモデルが会話の流れを理解できるように学習します。

2. モデルの選択 適切な言語モデルの選定:ChatRetrieverの構築には、事前学習済みのOpenAIのGPTやGoogleのBERTなどを選択しています。

3. コントラスト学習の適用コントラスト学習の設定:コントラスト学習は、モデルが正しい応答と不適切な応答を区別できるようにするための技術です。各会話において、正しい応答（ポジティブサンプル）と関連性の低い応答（ネガティブサンプル）を比較し、モデルがこの区別を学べるようにします。

4. セッションマスク付き指示チューニング（SIT）: 特殊トークンの使用:モデルの入力の最後に特殊トークン（例えば[MB]）を追加し、これを使ってテキスト全体の表現を捉えます。
指示チューニング:ユーザーの会話（セッション）とそれに対する応答を一緒にモデルに入力し、セッション部分をマスク（隠す）して、応答のみをモデルが生成できるようにします。これにより、モデルは会話の文脈をより深く理解することが強化され

5. モデルの訓練と評価 訓練: 上記の手法を使用してモデルを学習します。多くのエポックにわたって、モデルが会話の流れと適切な応答をどれだけ正確に生成できるかを改善していきます。多分プロンプトだと思ってる
評価: 実際の会話データを使ってモデルの性能を評価します。モデルがどれだけ正確に情報を検索し、適切な回答を生成できるかをチェックします。

6. 微調整と展開 微調整: 実際のユーザーのフィードバックを用いて、モデルをさらに微調整します。これにより、特定のユーザーや状況に合わせたパフォーマンスの向上が期待できます。
展開: モデルが十分に機能することを確認したら、実際のアプリケーションやサービスに組み込み、ユーザーが使用できるようにします。

結果

実験では、ChatRetrieverが他の検索システムよりも優れた結果を示しました。比較された他の検索システムには、いくつかの異なるアプローチが含まれています。これらは主に以下の三つのカテゴリに分類されます:

1. 会話型クエリ書き換え（Conversational Query Rewriting, CQR）: 会話型クエリ書き換えは、会話の文脈を単一の検索可能なクエリに変換する手法です。これにより、標準的な検索エンジンを使用して関連情報を検索できます。例としてT5QRやConvGQRがあります。
2. 会話型密集検索（Conversational Dense Retrieval, CDR）: このアプローチでは、会話全体を直接エンコードし、エンドツーエンドで密集した情報検索を行います。これにはConvDRやLeCoREなどが含まれます。
3. LLMベースの検索:このカテゴリには、大規模言語モデル（LLM）を活用して会話の文脈やクエリを理解し、適切な情報を検索するシステムが含まれます。例えばINSTRCUTORやLLM Embedderなどがあります。

ChatRetriever: Adapting Large Language Models for Generalized and Robust Conversational Dense Retrieval
ChatRetriever: 一般化された会話型検索のための大規模言語モデルの適応 2024

Figure 1には、大規模言語モデル(LLM)を会話型のクエリ書き換えと密集検索に適応させるプロセスが示されています。
この図は、会話型検索セッションからクエリを再構成し、それを利用して情報を密集して検索する概念を説明しています。

1. 会話型検索セッション(Conversational Search Session):ユーザーが対話形式で情報を求める場面。
例えば、ユーザーが「海の底は凍ることがあるか？」という質問をするシーンが示されています。

2. LLMによるクエリの再構成(LLM-based Rewriter):LLMが会話の文脈を受け取り、それを標準的な検索クエリに再構成します。
このプロセスを通じて、複雑な会話が単一の検索可能なクエリに変換される様子が描かれています。

3. 密集検索への適応(Adaption for Dense Retrieval):再構成されたクエリが情報検索のためにデータベースやインデックスに送信されるプロセス。
このステップでは、LLMが生成したクエリに基づいてデータベースから関連する情報が検索されます。
この過程で、LLMの一般化された検索能力を活用し、会話からのクエリに対して適切な情報を迅速に見つけ出すことができます。

ChatRetriever: Adapting Large Language Models for Generalized and Robust Conversational Dense Retrieval
ChatRetriever: 一般化された会話型検索のための大規模言語モデルの適応 2024

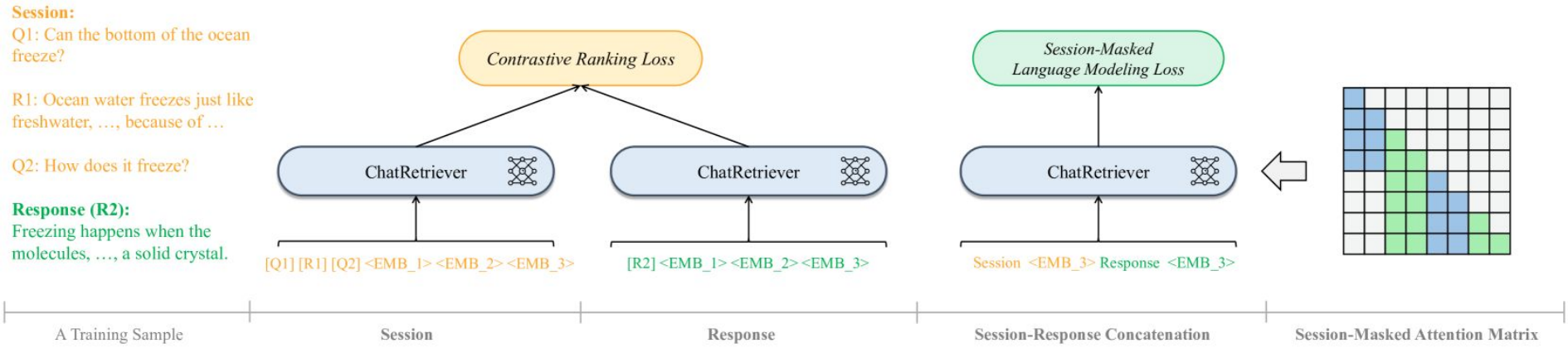


Figure 2 では、ChatRetrieverとしてLLMを微調整するプロセスが概説されています。
この図ではCSIT（Contrastive Session-Masked Instruction Tuning）による二重学習目的を用いた手法が示されています。

1. 特殊トークンの使用(Use of Special Tokens): 入力テキストの最後に特殊トークン（例えば、<EMB_3>）を使用します。
このトークンは、セッションまたは応答の全体を代表する役割を果たします。これにより、テキストの総合的な表現を効果的に捉えることができます。

2. セッションマスク付き注意マトリックス(Session-Masked Attention Matrix): 図中の青い四角はセッションまたは応答のトークンを表し、緑の四角はそれらの特殊トークンを示しています。このAttentionマトリックスは、モデルがセッションの文脈をどのように処理し、応答生成にどのように貢献するかを視覚的に示しています。

3. 二重学習目的の活用(Utilizing Dual Learning Objectives): CSITでは、コントラスト学習目的とセッションマスク付き指示チューニングを組み合わせます。これにより、モデルはセッションの複雑な文脈をより効果的に学習し、応答をより正確に生成する能力が強化されます。

ChatRetriever: Adapting Large Language Models for Generalized and Robust Conversational Dense Retrieval
ChatRetriever: 一般化された会話型検索のための大規模言語モデルの適応 2024

「C Prompts in Full Context Modification」のセクションでは、完全な文脈修正実験で合成された会話テキストを生成するためのプロンプトについて

図7で示されているプロンプトを使用して、ChatGPT3.5が生成した応答（緑色の内容）が示されています。
この実験では、元のクエリとその人間による修正版をLMに提示し、新しい文脈を生成するように求めます。
このプロセスは、会話の文脈を完全に変更することを目的としています。
提示されたプロンプトに基づき、ChatGPT3.5は新しい会話テキストを生成します。
このテキストは、変更された文脈に基づいて適切な応答や続きを提供することで、
会話が自然で一貫性があるかどうかを試すために使用されます。
以下がそのプロンプトです。

会話型クエリ、その文脈に依存しない書き換え、およびその応答が与えられた場合、
それに対して2ターンの会話型文脈を生成します。

このターン：
クエリ：それを修理するのにどれくらいかかりますか？
書き換え：ガレージドアオープナーの修理にはどれくらいかかりますか？
応答：ガレージドアオープナーの修理費用は、問題の範囲に応じて100ドルから300ドルの間で変動します。
トップに戻る。選択するガレージドアのタイプや、必要な追加部品や労力によって、
専門業者に依頼した場合の費用がどれくらいかかるかが影響されます。

合成された会話文脈：
クエリ1：新しいガレージドアオープナーのコストはどれくらいですか？
応答1：新しいガレージドアオープナーのコストは、ブランド、機能、および設置要件に応じて150ドルから500ドルの範囲です。
クエリ2：ガレージドアオープナーにはどのような一般的な問題がありますか？
応答2：ガレージドアオープナーの一般的な問題には、リモコン、モーター、センサー、またはドア自体の問題が含まれます。

Given a conversational query, its context-independent rewrite, and its response, generate *two* turns of conversational context for it.

This turn:
 # Query: *How much does it cost for someone to fix it?*
 # Rewrite: *How much does it cost for someone to repair a garage door opener?*
 # Response: *Garage door opener repair can cost between $100 and $300 depending on the extent of the problem. Return to Top. The type of garage door you select -- and any extra pieces or labor required -- will influence how much you pay to have it professionally…*

Synthetic Conversation Context:
Query1: How much does a new garage door opener cost?
Response1: The cost of a new garage door opener can range from $150 to $500, depending on the brand, features, and installation requirements.

Query2: What are some common problems with garage door openers?
Response2: Some common problems with garage door openers include issues with the remote control, the motor, the sensors, or the door itself.

CONQRR: Conversational Query Rewriting for Retrieval with Reinforcement Learning 会話クエリ改稿による検索のためのCONQRR: 強化学習を用いたアプローチ 2022

概要

CONQRRは、話している内容に基づいて質問を書き換えることで、情報を見つけるための既存の検索ツール（リトリーバー）をより上手に使えるようにします。特に、会話の流れの中で質問がどのように変わるかを学び取り、それに応じて最適な検索を行うことができます。

手法

CONQRRは、会話型クエリ改稿（Conversational Query Rewriting）のためのモデルで、強化学習を用いて最適化されています。以下のステップで進めることができます：

1. 問題の定義:会話の文脈（過去のやり取り）と現在のユーザーの質問を入力とします。目的は、入力された会話の文脈と質問から、検索エンジンが理解しやすい形の独立した質問へと書き換えることです。

2. データの準備:会話型質問応答データセット（例えばQReCCなど）を用意します。各会話には、元の質問とそれに対応する書き換えられた質問（ラベル）が含まれている必要があります。

3. モデルの選択と設定:T5などのSeq2Seqモデルをベースとして使用します。このモデルを事前に大量のテキストデータで事前学習させたものを使用すると、学習が効率的に進みます。

4. 監督学習での事前学習:会話の文脈と現在の質問を入力とし、正しい書き換えられた質問を出力するようにモデルを訓練します。クロスエントロピー損失を使用して、モデルが正しい書き換えを生成できるようにします。

5. 強化学習による微調整:既に監督学習である程度訓練されたモデルを用いて、さらに強化学習を行います。報酬関数を定義して、検索性能を直接向上させる方向にモデルを最適化します。この報酬は、書き換えられた質問を使って検索を行い、その結果がどれだけ良いか（例えば、適切な情報をどれだけ取得できたか）に基づいています。

6. 報酬の計算:き換えたクエリを使って実際に情報検索を行い、その結果から報酬を計算します。検索結果が良好であれば高い報酬を、悪ければ低い報酬を与えることで、モデルがより適切なクエリ改稿を学ぶよう促します。

7. 評価と調整:モデルのパフォーマンスを評価するために、様々なメトリクス（例えば、精度、リコールF1スコアなど）を用いて評価します。不足している部分や改善が必要な点に基づいて、モデルの調整を行います。

8. デプロイメント: モデルの学習が完了したら、実際のアプリケーションに組み込み、リアルタイムでの会話型クエリの書き換えに利用します。

結果

完全に新しい会話に対しても、良い検索結果を出せるっぽい
会話の流れに強く依存するため、文脈が薄い場合や短い会話ではうまく動かなさそう

Appendix