論文要約

LLM関連

LONG2RAG: Evaluating Long-Context & Long-Form Retrieval-Augmented Generation with Key Point Recall LONG2RAG: 長文コンテキストおよび長文形式の検索強化生成の評価とキーポイントリコールによる評価

概要: LONG2RAGは、長文コンテキストでの LLMのRAG性能を評価するためのベンチマークです。 280の質問に5つの関連文書を設定し、検索された文書から抽出されたキーポイントをどれだけ含んでいるかを測定するKPRで評価。GPT-4oが最高スコアの 0.579を記録

RAG評価で主に長文コンテキストに対応する LLM評価ベンチマーク LONG2RAGを提案。280の質問が10の領域で設定され各質問に対して 5つの関連文書を設定。評価指標には KPR（Key Point Recall）を設定し検索された文書から抽出されたキーポイントをどれだけ含んでいるかを測定する方法（各スコアは 0から1の範囲で、高いほど良い性能を示します）を使用。質問は 8つのカテゴリ（事実、説明、比較、主観、因果関係、仮定、予測、方法論）に分類して評価、 GPT-4oのKPRは 0.579、Claude-3-SonnetのKPRは 0.477、Qwen2-72B（オープンソースモデルの大規模版） : KPRは、0.449、Phi-3-mini-128K: KPRは 0.434と商用モデルである GPT-4oが最も優れた結果を示しました。

また、KPRは長文生成を好む傾向もあるため、生成の質と長さのバランスが重要であることもわかっています

技術や手法

- **LONG2RAGベンチマーク **: 280の質問を使用し、各質問に対して平均 2444語の5つの検索文書が関連付けられています。これにより、モデルが長文の検索情報を取り込む能力を評価します。

- **キーポイントリコール（ KPR）**: 検索された文書から抽出されたキーポイントが生成された回答にどの程度含まれているかを評価する手法です。この評価を通じて、モデルが検索情報を活用しているかどうかを測定します。

- **データセットの構築方法 **: 自動パイプラインを用いて質問を生成し、関連する文書を検索してキーポイントを抽出。その後、 LLMと人間の協力によりキーとなるポイントの検証を行い、データセットを構築しました。

評価手法とパフォーマンス指標

論文では、LONG2RAGベンチマークを用いて 9つの最新の LLM（大規模言語モデル）を評価しました。評価に用いられた指標は以下の通りです。

Benchmarking Large Language Models in Retrieval-Augmented Generation 検索強化生成における大規模言語モデルのベンチマーク

概要: RAGのパフォーマンスを 4つのノイズ耐性、否定拒否、情報統合、反事実耐性の　RGBのコーパスで評価。LLMは一定のノイズ耐性を持つが、否定拒否や情報統合、誤情報処理にはまだ課題が多いことがわかりました

技術や手法

- **検索強化生成 (Retrieval-Augmented Generation, RAG):** RAGは、検索エンジンを用いて外部の知識を取得し、モデルの幻覚を軽減する手法。特に、インターネット上の膨大な情報から正確な知識を得るために使用される。

- **Retrieval-Augmented Generation Benchmark (RGB):** RGBは、RAGの4つの基本的な能力を評価するために設計された新しいベンチマークで、最新のニュース情報を基に構築されている。このベンチマークにより、LLMがノイズ情報に対してどの程度頑健であるかや、複数の情報を統合する能力などを評価できる。

    - **ノイズ耐性 (Noise Robustness):** 質問と関連があるが、回答を含まないノイズ文書から必要な情報を抽出する能力。

    - **否定拒否 (Negative Rejection):** 必要な知識が取得された文書に存在しない場合に、適切に回答を拒否する能力。

    - **情報統合 (Information Integration):** 複数の文書から情報を統合して質問に回答する能力。

    - **反事実耐性 (Counterfactual Robustness):** 取得された文書に誤った情報が含まれている場合に、そのリスクを認識して適切に処理する能力。

使用用途

この研究は、以下のようなシーンで活用が期待される:

- **検索エンジンの改善 :** LLMを用いた検索結果の生成において、ノイズ情報を適切にフィルタリングし、より正確な情報提供を行う。

Investigating the Role of Prompting and External Tools in Hallucination Rates of Large Language Models プロンプト設計および外部ツールの役割が大規模言語モデルの幻覚率に与える影響の調査

概要: LLMのハルシネーションがプロンプトエンジニアリングや　LLMエージェントの活用でどのように変わるかを調査

Temperatureをあげて複数回の LLM呼び出しの多数決で回答する SCを使用することが効果的だという結果になり、現実での知識を問うタスクには　KGRが効果的という結果になりました

技術や手法

1. プロンプト技術

1.1 チェイン・オブ・ソート（ CoT）プロンプト

チェイン・オブ・ソート（ Chain-of-Thought, CoT）プロンプトは、複雑な問題をより簡単に解決できるように、小さなステップに分割する手法です。この方法では、モデルが一度に問題全体を解決するのではなく、解決の過程を段階的に分解します。例えば、数学の問題を解く場合、問題をいくつかの小さなステップに分けて、それぞれのステップで部分的な答えを導き出し、最終的に全体の答えに到達します。この方法により、LLMはより精度の高い推論が可能になります。

1.2 自己一貫性（ SC）

自己一貫性（ Self-Consistency, SC）は、同じ質問に対して複数回の LLM呼び出しを行い、その結果を多数決で選ぶことで一貫性のある答えを導き出す手法です。この手法の目的は、モデルのランダムな生成によって生じる不安定な出力を安定させることです。温度（ temperature）の設定を調整し、複数の異なる出力から最も一貫した答えを選ぶことで、信頼性の高い回答が得られるようになります。この方法は、特に数学の問題や論理的な推論を必要とする課題に有効です。

1.3 木構造の思考（ ToT）

Distance between Relevant Information Pieces Causes Bias in Long-Context LLMs 関連情報の間の距離が長コンテキスト LLMにバイアスを引き起こす

概要: 長文を扱うLLMには、情報が中間にあると見落とす lost in the middle問題の他に複数の情報を活用して回答するときにその複数の情報同士の距離とその配置が遠くなることが結果に影響することが開発されたLONGPIBENCHというベンチマークからわかりました

技術や手法

1. **ポジショナルバイアスの問題と「 lost in the middle」現象 **

- **ポジショナルバイアス **とは、大規模言語モデル(LLM)が入力された情報の位置に応じて、その情報をうまく扱えなくなる現象を指します。この論文では、特に長文の入力での問題を扱っています。具体的には、重要な情報が文脈の中間に位置する場合、モデルがその情報を見落とす「 lost in the middle」現象が問題視されています。この現象は、 LLMsが長い文脈を効率的に利用する際の大きな障害です。

2. **LONGPIBENCHの設計と目的 **

- **LONGPIBENCH**は、複数の関連情報が含まれるタスクにおけるポジショナルバイアスを評価するためのベンチマークです。このベンチマークは、絶対位置と相対位置のバイアスを評価することを目的としています。

  - **絶対位置 **とは、文脈全体の中で関連情報がどの部分に位置するかを指します(例えば、入力の先頭、中間、末尾など)。

  - **相対位置 **は、複数の関連情報の間の距離や、それらの情報がどの程度密集しているかを意味します。この点に注目することで、 LLMが情報の分布や配置にどのようなバイアスを持っているかを評価します。

3. **LONGPIBENCHのタスク設計 **

DIVERSITY HELPS JAILBREAK LARGE LANGUAGE MODELS 多様性が大規模言語モデルの脱獄を支援する

概要: LLMの脱獄手法を試し GPT-4、Gemini、Llamaなどの主流のチャットボットに対して成功率を最大　62%向上、元の攻撃プロンプトから大きく逸脱した表現を使う多様化ステップ（　Diversification Step）と元の攻撃意図を保持しつつ、それを曖昧化するためのプロンプト隠蔽ステップ（　Obfuscation Step）の2ステップを使う DAGRフレームワークを提案

多様な攻撃手法の概要

本論文では、多様な攻撃手法を使用して大規模言語モデル（　LLM）を脱獄させることで、安全性を突破する成功率を向上させています。この多様な攻撃手法は以下のように　2つのステップから構成されており、それぞれの役割がモデルの脆弱性を効果的に突きます。

1. **多様化ステップ（ Diversification Step）**

2. **隠蔽ステップ（ Obfuscation Step）**

1. 多様化ステップ（ Diversification Step）

- **攻撃の創造性を最大限に活用する **: 多様化ステップでは、以前に生成された攻撃プロンプトから大きく逸脱し、過去の攻撃手法と異なる新たな攻撃プロンプトを生成することを目指します。この多様化には創造性やフィクションを取り入れており、攻撃手法の幅を広げることを重視しています。

- **具体的な生成方法 **:

    - 各深度において、新しい攻撃プロンプトを生成する際に、創造的でフィクションを含んだ内容にするように攻撃モデルに指示されます。

    - 攻撃対象の言語モデルを破るために、物語風の設定や仮想のシナリオを使用して、安全性メカニズムが警戒しにくい方法で攻撃を試みます。

RAGULATOR: LIGHTWEIGHT OUT-OF-CONTEXT DETECTORS FOR GROUNDED TEXT GENERATION RAGULATOR: 文脈外検出器による軽量なグラウンド化されたテキスト生成

概要: RAGulatorは、LLMがRAGで生成した文と、関連する文脈を比較することで外れていないかを　BERTベースの軽量モデルを使い、生成された文がその文脈と一致しているかどうかを分類します。このモデルは、要約や類似性データを基に訓練され、文脈外（　Out-Of-Context, OOC）の可能性がある部分を識別します。文が文脈から外れていると判断された場合、それは信頼性の低い情報、つまり「文脈外」とみなされます。

RAGulatorは、RAGにおいて、AIが生成する文が文脈に沿っているかどうか、つまり「文脈外」　(Out-Of-Context, OOC)の検出を目的としています。 RAGは事前に検索された情報を元に回答を生成しますが、その際に文脈から外れた内容を生成してしまうこと（いわゆる「幻覚」や「ハルシネーション」）が問題となります。このような不正確な情報は特に企業環境では深刻な影響を及ぼす可能性があります。

そこで、RAGulatorは軽量な分類器を用いて、生成された文が文脈に基づいているかどうかを判別する仕組みを提供しています。

2. データセットの生成

**データセットの準備 **は、RAGulatorの重要なステップです。このデータセットは、要約データセットとセマンティックテキスト類似性（　Semantic Textual Similarity, STS）データセットを活用して構築されました。

- **要約データセット **（BBC, CNN/Daily Mail, PubMed）:

  - 各データセットは要約と元記事のペアで構成されています。要約と無関係な記事をペアリングし、それぞれの要約を文に分割して、文脈外（　OOC）の例を生成します。これにより、文脈に沿っている場合と沿っていない場合を含む学習用データセットを作り出しています。

- **セマンティックテキスト類似性データセット　**（MRPC, SNLI）:

  - 文ペアとその類似性ラベルを元に、文脈内および文脈外の文ペアを作成。例えば、ランダムな文を追加して文脈を拡張し、それらが類似するかどうかで文脈内・外のラベルを付けました。

3. データセットの整形とラベリング

RAGVIZ: Diagnose and Visualize Retrieval-Augmented Generation RAGVIZ: リトリーバル・オーグメンテッド・ジェネレーションの診断と可視化

https://github.com/cxcscmu/RAGViz

https://www.youtube.com/watch?v=cTAbuTu6ur4

概要: RAGVizはモデルがどの文章に注目して答えを生成したかを各文書やトークンにどれくらい注意しているかをクエリとキーの内積をソフトマックス関数で正規化して可視化、どの文章が生成の根拠かをわかります

技術や手法

1. **注意可視化 **:
   - 取得した文書に対するトークンの注意度を可視化する機能を提供します。生成されたトークンが取得文書のどの部分にどの程度依存しているかを視覚的に確認できます。
2. **文書トグル機能 **:
   - 取得した文書を追加または削除して生成の変化を比較できる機能を持っています。これにより、どの文書が生成結果にどのように影響するのかを容易に評価できます。
3. **分散アーキテクチャ **:
   - RAGVizは、複数のノードにデータセットを分割して効率的に処理を行います。また、効率的なLLM推論ライブラリを使用し、低遅延でのLLM出力生成を実現しています。
4. **スニッピング技術 **:
   - ドキュメントの一部を抜き出して使用するスニッピング技術を実装しており、クエリに最も関連する部分を取得するためにスライディングウィンドウを利用するなどの最適化を行っています。

注意スコアの算出方法は、トランスフォーマーモデルの「アテンション機構」を使用します。具体的には、以下のステップで注意スコアが計算されます。

AIOS Compiler: LLM as Interpreter for Natural Language Programming and Flow Programming of AI Agents AIOSコンパイラ：自然言語プログラミングと AIエージェントのフロープログラミングにおける　LLMを通訳者として使用

https://github.com/agiresearch/OpenAGI

https://github.com/agiresearch/AIOS

概要：Code Representation and Execution (CoRE)というAIエージェントを自然言語で「ステップ名」、「ステップタイプ」、「ステップ指示」、「ステップ接続」という　　4つの基本構成で整理しながら途中のステップで生成した結果や中間データを保存する外部メモリと、ツール（例えば、検索エンジンや　　API呼び出しツール）を利用できる環境を構築します

1. システムの全体構造

- **LLMのインタプリタとしての利用　**LLM（例: GPT-4やMixtralなど）を中心に据え、ユーザーの入力（自然言語プログラム）を解釈し、次のアクションを決定するコンポーネントを開発します。

- **外部メモリとツールの連携　**LLMが途中のステップで生成した結果や中間データを保存する外部メモリと、ツール（例えば、検索エンジンや　　API呼び出しツール）を利用できる環境を構築します。

- **CoRE言語を用いたプログラム表現　**自然言語、疑似コード、フロープログラミングを統合した形式でプログラムを表現するための独自のシンタックス（例えば、「　　:::」を使ったステップの構造化）を設計します。

2. 実装ステップ

ステップ 1: CoRE言語の定義とパーサーの実装

- **CoRE言語の構文定義 **CoRE言語の基本構造を定義します。各ステップには「ステップ名」「ステップタイプ（処理、判断、終了）」「指示」「次のステップへの接続」が含まれます。

- **構文解析（パーサー）　**CoRE言語で記述された自然言語プログラムを解析し、それを理解可能なデータ構造に変換するためのパーサーを実装します。　　Pythonであれば正規表現や文法解析ツール（例　：`lark-parser`や`PLY`）を利用して実装可能です。

Foundations and Recent Trends in Multimodal Mobile Agents: A Survey マルチモーダル移動エージェントの基礎と最近の動向　: 調査

https://github.com/aialt/awesome-mobile-agents

概要: モバイルエージェント技術の紹介。 LLMを使い、ユーザーの指示に従って画面操作を自動化。視覚情報とテキスト情報を統合することで、ボタンの押下や画面のナビゲーションを実施

技術や手法

1. **プロンプトベースの手法 **

　大規模言語モデル(LLM)を活用し、指示に基づいてタスクを実行するアプローチです。特に、 Chain-of-Thought(CoT)推論などを用いて、GUIの操作をより効率的に行えるようになっています。代表例には、OmniActやAppAgentがあります。

2. **訓練ベースの手法 **

　マルチモーダルモデルをモバイル環境向けに微調整するアプローチです。特に、 LLaVAやLlamaなどのモデルが用いられ、視覚データとテキストデータの統合処理を行うことにより、インターフェースのナビゲーションやタスクの実行が可能となっています。

3. **補完技術 **

　エージェントのパフォーマンスを向上させるための技術として、視覚エンコーダーの改善やモバイル特有のインタラクティブ要素を強化するデータセットの導入が挙げられます。

An Empirical Study on the Potential of LLMs in Automated Software Refactoring LLMの自動ソフトウェアリファクタリングにおける可能性の実証的研究

概要: LLMによる自動リファクタリングするためにリファクタリング検出ツール　RefactoringMirrorを提案。

ChatGPTとGeminiで実施したリファクタリングを検出し、 IntelliJ IDEAのAPIを利用して再適用することで自動での成功率が向上しています

技術や手法

1. **LLMによるリファクタリング機会の特定　**:

　- Javaプロジェクトから 180のリファクタリングケースを収集し、 ChatGPTおよびGeminiモデルを利用してリファクタリング機会を特定。

　- 一般的なプロンプトに加え、具体的なリファクタリングタイプを指定したプロンプトを使用することで、　LLMの成功率を改善。

2. **リファクタリングソリューションの提案　**:

　- ChatGPTは180のリファクタリングに対して 176のソリューションを提案し、そのうち 63.6%が専門家と同等、またはそれ以上の品質であると評価された。

3. **RefactoringMirror手法**:

　- LLMが生成したリファクタリングの結果に対し、 ReExtractorツールを用いてリファクタリングの詳細を検出し、それを　IntelliJ IDEAなどの確立されたリファクタリングエンジンで再適用することで、安全性を向上させる方法です。

　- この手法により、LLMが提案したバグのあるリファクタリングの全てを避けることに成功しました。

使用用途

- **自動コード改善**:

Appendix