大規模言語モデルを用いた日本語文中の並列構造の抽出

島森瑛貴
株式会社ゆめみ e_shimamori@yumemi.co.jp

概要

本稿では、大規模言語モデルを用いて日本語文中の並列構造を抽出した。提示したテキスト中の並列構造を抽出するプロンプトを作成し、slack のチャットテキストを利用して評価した。オープンテストの結果、F1 値はベースラインで 0.2036、提案手法で 0.3693 となり、適切なプロンプトを選択することで抽出の精度が向上することが明らかとなった。また、チャットテキストとはスタイルの異なる書き言葉のテキストについても評価し、スタイル毎の精度の変化を評価した。実験の結果、F1 値はチャットテキストで 0.3693、書き言葉で 0.2129 となった。

1 はじめに

近年リモートワークやチャットで業務連絡を行う機会が増加している。しかし実際に出社する場合と比較すると、業務以外の雑談の頻度が低下するため他者の人間性が伝わりにくく、コミュニケーションが取りにくくなる危険性がある。チーム開発の生産性向上のためには適切にコミュニケーションをとることが重要である。そこで個人の興味関心の関連に基づき相性の良い人同士を案件に配属することで、コミュニケーションが円滑になり生産性が向上するという仮説を立てた。

ゆめみ社では各社員が自分の興味関心を社内に発信するチャンネルが slack に用意され業務外の内容も発信できる環境が整備されている。これらの蓄積されたデータを分析することで、社員ごとの興味関心の関連性を把握することができると考えている。

本稿では手始めに、社員が興味関心を持つ関連性のある複数の語として、チャットテキストに含まれる並列関係にある名詞句を LLM を使用して抽出するようなプロンプトを作成し、その評価を行う。

今回対象とするテキストチャットは、一般的な書き言葉のテキストと比べると文法性が低く解析の難易度が増し既存の評価データや解析手法ではうまく評価できないと考えられる。そこで、このような解析が難しい文に対して自由度が高い方法で解析するために大規模言語モデルを用いる。また書き言葉のテキストも同様のプロンプトで評価し、スタイルの違いが並列構造の抽出の精度にどのように影響するかを検討する。

評価の結果、提案したプロンプトの精度がベースラインと比較して 16 ポイントほど改善した。また、書き言葉のテキストと比較してチャットテキストに対して特に有効であることが明らかとなった。

2 関連研究

2.1 日本語文中の並列構造の検出

文中から並列構造を検出するこれまでの研究 [1, 2, 3] では、「と」や「や」のような並列助詞、「または」や「かつ」のような接続詞を含む文節を並列キー [1] として検出し、その並列キーの前後で詳細な範囲を推定し、並列要素を特定する。

このタスクの大きな課題として、並列構造の範囲推定が困難であるということが挙げられる。例えば文中に「A の B と C の D」という部分文字列があるとする。「B と」を並列キーとみなした場合、この部分文字列の範囲外に並列の要素が存在しないと仮定しても、「A の B」と「C」、「A の B」と「C の D」、「B」と「C」、「B」と「C の D」の 4 通りが並列構造の要素の組として考えられる。並列構造を構成する要素は構造や語に類似性が認められることが多いため、要素の範囲を推定する手掛かりとしてこれらを利用することが多い。しかし、このようなミクロな特徴のみを用いた推定では不十分なことが多い。

その大きな原因として「語の類似度」の評価が難しいという問題が挙げられる。語の類似性は直接的類似性と間接的類似性の二つに大きく分けられる [4]。このうち、直接的類似性は対象の語がどちらもシソーラスに登録されていれば使用することができるが、未登録の語が出現する場合や、句以上の単位を比較する場合には類似性の評価が困難である。また間接的類似性については、類似であるかどうか

が文脈に大きく依存し、適切な概念ベースを選択する必要があるという課題がある。これらを踏まえると、並列構造を適切に選択するためには表層だけではなく、適切に意味を理解する必要がある。

2.2 LLM のプロンプト

近年は大規模言語モデル (LLM) を用いてさまざまな自然言語処理のタスクを解くという試みが行われている。LLM を利用して構造データを解析できるという先行研究 [5, 6] もあることから、LLM を利用することで、ミクロな規則を作成することなく適切に並列構造を取得できると期待できる。

代表的なプロンプト手法として Few-shot Learning[7],Generate Knowledge Prompting(GKP)[8],Chain-of-Thought Prompting(CoT)[9] などが存在する。

Few-Shot Learning は指示だけでなく具体例を併せて提示する手法である。指示のみで具体例を提示しないものは Zero-Shot、1 つだけ例を提示するものは One-Shot、いくつかの例を提示するものは Few-Shot と呼ばれる。

Generate Knowledge Prompting は、問題を解く上で必要な前提知識をプロンプトで与える手法である。例えば、今回のタスクであれば、「並列構造の定義」や「並列構造にはどのような特徴が存在するか」などを情報として明示しておくことに相当する。

Chain-of-Thought Prompting は最終的な答えだけではなくそこに至るまでの過程を考えさせる手法である。1-shot や few-shot であれば、模範解答を示す際に、最終的な答えだけではなくそこに至るまでの思考も併記することであり、0-shot であれば、「複数のステップに分けて考えてみましょう」などの文言をプロンプトに追加することに相当する。

また、「優れているので自信を持ってください」や「困難なタスクですがチャレンジしてみましょう」など感情に訴えかける [10],「タスクに取り組む前に一度深呼吸をしましょう」など落ち着かせる [11] ようなプロンプトを追加することでもタスクの精度が向上したという研究結果も存在する。

そのほかにも、質問の内容や出力のスタイルを明確にする、プロンプトを構造化する、指示と文脈を「#」などの文字で明確に区切る、「あなたは自然言語処理の研究に関わる専門家です」のようにロールを設定するなどの tips[12] を利用する。

本稿ではこれらの先行研究を踏まえて、LLM を利用して文章中の並列構造を抽出するプロンプトを作成し、その評価を行う。

3 提案方針

先行研究 [8, 9, 10, 11] で提案されている手法をもとにプロンプトを作成し、これらを評価する。

ベースラインとして指示と出力形式だけを指定する 0-shot のプロンプト (A.1) を用意する。これを基本として以下の指示、例は「# 出力形式」と「# 対象」の間に追加する。具体的なプロンプトは後ろの付録に追加する。また簡単のため共通部分は省略する。

3.1 具体例

文中に並列組が 1 つ存在する場合、存在しない場合、2 つ以上存在する場合のそれぞれで正しい出力例を表示する (A.2)。

3.2 原文に含まれていない出力を禁止

LLM は出力を確率的に生成するため、抽出タスクの際に、元の文に含まれていない文字列を解答として出力することがある。そのため、元の文に含まれていない文字列を解答として出力することをプロンプトで明示的に禁止する (A.3)。

3.3 並列構造の知識を生成する

Generate Knowledge Prompting の手法を適用して、並列構造がどのようなものであるかの定義や具体的な特徴をプロンプトに明記する (A.4)。

今回対象とする名詞の並列構造については明確な定義が見つけられなかったため著者が定義したものを、また並列構造の特徴については [13] の内容を改変して利用した。具体的な改変内容は平易化、文体の統一、例文の説明の具体化、並列キーの手がかりを明記する意図で「並列キーとしては」で始まる一文の追加である。

3.4 元の部分文字列の抽出を強制

slack では markdown 記法が使えるため、元のテキストには「`if` 文」のように記法に起因する記号が付いていることがある。抽出する際にこのような記号が勝手に削除されないように、元の文中の部分文字列をそのまま出力するように明記する (A.5)。

3.5 助詞の補完

今回対象とする slack のチャットの内容は、話し言葉の書き起こしとは異なり言い直しやフィラーは含まれないが、livedoor ニュースコーパスなどの web 記事と比較すると文体が砕けているものが多い。そのため、必ずしも文法的に正しいとは限らず、このような砕けた文体が原因で解析を誤り、精度が低下することが考えられる。そこで、文を正規化することでそのような精度低下を防ぐことを検討する。今回のデータセットに顕著な特徴として、必要な助詞が欠落しているというものがある。そこで欠落した助詞を一度適切に補完し、その後並列構造を抽出する。このとき、追加した助詞が出力されると元の文の部分文字列と一致しないため、このステップで追加した助詞は出力に含めないように明記する (A.6)。

3.6 感情、ロール、深呼吸

また、そのほかに、感情に訴えかける指示、役割を指定する指示、深呼吸を促す指示 (A.7) をそれぞれプロンプトに追加する。

3.7 Chain-of-Thought Prompting

Chain-of-Thought Prompting の手法を適用して、思考の過程を段階的にプロンプトに明記する。並列構造が存在する例文の場合は、並列キーを明示しその前後の並列要素とその根拠をプロンプトに追加する。並列構造が存在しない例文の場合は並列キーが存在しないため並列構造が存在しないということをプロンプトに追加する。Zero-Shot の場合は、「段階的に考えろ」というプロンプトを追加する (A.8)。これらは各例文のクエリと出力の間に追加する。

4 評価

実験設定 3 章で述べたプロンプトを評価する。ベースライン (BL) としてタスクの説明と出力の方針のみを指定したプロンプト (A.1) を利用する。それに加え、A.2 〜A.8 をそれぞれ追加したもの、これら全ての指示のみを追加したもの (BL+すべての指示)、これら全ての具体例だけ追加したもの (BL+すべての具体例)、これら全ての具体例とその思考過程を追加したもの (BL+すべての具体例+CoT)、全ての指示・具体例を追加したもの (BL+全部) の合計 14 個について、システムの推定結果と正解が一致するか適合率、再現率、F1 値を求めた。LLM は temperature と top_p はともに 0.0、seed は 0 として OpenAI の gpt-3.5-turbo-1106 を使用した。

データセット 評価の対象は、ゆめみ社の slack の 2020 年の全投稿のうち、品詞が並立助詞である形態素を含む文を 400 件ランダムに抽出した。対象のうち 100 文をシステム改良のための分析に利用し、クローズドテストとして評価する。残りの 300 文を未参照の事例に対する評価を行うときのデータとして使用し、オープンテストとして評価する。並立助詞が含まれるかを判断するための形態素解析器には、MeCab[14] をデフォルトの IPA 辞書で使用した。

並列要素の正解は著者がアノテーションを行った。前述したように名詞句の並列のみを対象とし、「〜し、〜した」のような節の並列は対象としない。クローズドテストの 100 文には並列要素は 107 件存在し、オープンテストの 300 文中には並列要素は 242 件存在した。

クローズドテストの評価を表 1 に、オープンテストの評価を表 2 に示す。

スタイルごとの比較 今回我々が対象としているチャットテキストは、書き言葉ではあるものの、言葉遣いは砕けており、内容があまり修正されていないため文法的に正しくないものが含まれるという点で、話し言葉の書き起こしと書き言葉の中間のようなスタイルである。今回提案する並列構造の抽出手法がこれらのスタイルの違いに対してどのように影響するかを調査するために、14 種類のプロンプトのうちクローズドテストでもっとも精度が F1 値が高かった、ベースラインに全てを追加したプロンプトで、書き言葉のテキストを評価する。

書き言葉のテキストとして livedoor ニュースコーパスのスポーツ記事を利用する。いずれも上記の実験と同様に、並立助詞を含む 300 文で適合率、再現率、F1 値を求めた。この 300 文中に並列構造は 258 件存在した。正解のアノテーションは著者が行なった。この実験の結果を表 3 に示す。

5 考察

個別の追加プロンプトでは検出精度が低下したものも存在したが、それらをすべて組み合わせた提案手法では精度がベースラインから大幅に改善した。ベースラインに 1 つ追加しただけでは特に大幅な精度向上が見られるプロンプトは存在しなかった。これは、追加した部分がそれぞれ箇所の改善に寄与す

表 1 クローズドテスト

	再現率	適合率	F1 値
ベースライン (BL)	0.3177	0.1650	0.2172
BL+0-shot CoT	0.2990	0.1658	0.2133
BL+具体例	0.3738	0.2162	0.2739
BL+抽出強制	0.3551	0.1751	0.2345
BL+知識生成	0.3644	0.1673	0.2294
BL+記号削除禁止	0.1962	0.1381	0.1621
BL+助詞の補完	0.3364	0.1578	0.2149
BL+感情	0.2710	0.1858	0.2205
BL+役割	0.2429	0.1287	0.1682
BL+深呼吸	0.2149	0.1703	0.1900
BL+すべての具体例	0.4859	0.3040	0.3741
BL+すべての具体例+CoT	0.4485	0.3096	0.3664
BL+すべての指示	0.3551	0.1919	0.2491
BL+全部	0.4392	0.3790	0.4069

表 2 オープンテスト

	再現率	適合率	F1 値
ベースライン (BL)	0.3933	0.1374	0.2036
BL+0-shot CoT	0.3891	0.1417	0.2078
BL+具体例	0.4016	0.2324	0.2944
BL+抽出強制	0.4453	0.1270	0.1977
BL+知識生成	0.3807	0.0931	0.1496
BL+記号削除禁止	0.2605	0.1618	0.1996
BL+助詞の補完	0.4369	0.1580	0.2321
BL+感情	0.4142	0.0873	0.1442
BL+役割	0.3807	0.06026	0.1040
BL+深呼吸	0.3723	0.1120	0.1723
BL+すべての具体例	0.4811	0.2072	0.2896
BL+すべての具体例+CoT	0.5210	0.2666	0.3527
BL+すべての指示	0.3430	0.1184	0.1761
BL+全部	0.4493	0.3134	0.3693

るためと考えられる。また、オープンテストにおいて適合率、再現率がともにベースラインより低下した「知識生成」、「役割」、「深呼吸」を除いたプロンプトで評価したところ F1 値が 0.3603 となり「BL+全部」より低かったことから、単体で精度が向上したプロンプトのみを利用することは必ずしも全体の精度向上にはつながらないことが明らかとなった。

また、チャットテキストの方が書き言葉のテキストより抽出の精度が高いが、これは一文の長さが短い方が解析の精度が高くなるためと考えられる。

5.1 プロンプトの違い

「BL+深呼吸」や「BL+知識生成」、「BL+感情」などの Zero-Shot で抽出させるプロンプトでは、抽出の対象となるテキスト以外のプロンプトの指示の部分が並列構造として出力されているものがあった。例えば、「BL+感情」では「' 困難なタスク: あなたは非常に優れているので自信を持ってチャレンジしてみましょう'」のようなものが並列構造としてシステムから出力されている。この原因として、先頭の指示文が「以下の文章中に含まれる並列要素の組をすべて漏らさずに列挙してください。」であることが考えられる。この指示の部分も「以下の文章中」であり、正解の例も提示していないため、システムが誤って解釈したと考えられる。そこで、「以下の文章中に」を「「対象」の「クエリ:」以降のテキスト」のように具体的に明示したところ、このような誤検出は減少したが、F1 値自体は変更前より低下した。

表 3 テキストのスタイルごとの違い

	再現率	適合率	F1 値
slack	0.4493	0.3134	0.3693
livedoor	0.3333	0.1564	0.2129

5.2 対象の文のスタイルの違い

一文あたりの平均文字数はクローズドテストに使用したデータでは 29 文字、オープンテストでは 77 文字、livedoor コーパスでは 128 文字であり、対象の文が長いほど抽出の精度が低下することが考えられる。名詞句の並列のみを対象とする場合、節境界を超えて並列が出現することは稀なため、節の単位で文を分割することで対象の文字数が減少し、抽出の精度が向上すると考えられる。

6 まとめ

本稿では LLM を使用して文章中の並列構造を抽出する手法について検討した。また、文書のスタイルの違いで抽出精度にどのような違いがあるのかについて検討した。今回は「並立助詞」を含む文のみを評価対象としていたが、実際には並立助詞を含まずに、読点や中黒を伴うものや、何も手がかりを伴わない名詞の列挙などが存在する。このようなものに対する精度向上は今後の課題である。また、プロンプトの内容自体は同一でも、記述する場所を変更すると抽出の精度が変わることがあり、このようなプロンプトの最適化も今後の課題である。

参考文献

[1] 首藤公昭, 吉村賢治, 津田健蔵ほか. 日本語技術文における並列構造. 情報処理学会論文誌, Vol. 27, No. 2, pp. 183–190, 1986.

[2] 黒橋禎夫, 長尾真ほか. 長い日本語文における並列構造の推定. 情報処理学会論文誌, Vol. 33, No. 8, pp. 1022–1031, 1992.

[3] 山腰貴大, 大野誠寛, 小川泰弘, 中村誠, 外山勝彦ほか. ニューラル言語モデルを用いた法令文の並列構造解析とその評価. 研究報告自然言語処理 (NL), Vol. 2017, No. 19, pp. 1–10, 2017.

[4] 渡部広一, 河岡司. 常識的判断のための概念間の関連度評価モデル. 自然言語処理, Vol. 8, No. 2, pp. 39–54, 2001.

[5] Tingyu Xie, Qi Li, Jian Zhang, Yan Zhang, Zuozhu Liu, and Hongwei Wang. Empirical study of zero-shot ner with chatgpt. **arXiv preprint arXiv:2310.10035**, 2023.

[6] Xuefeng Bai, Jialong Wu, Yulong Chen, Zhongqing Wang, and Yue Zhang. Constituency parsing using llms, 2023.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. **Advances in neural information processing systems**, Vol. 33, pp. 1877–1901, 2020.

[8] Jiacheng Liu, Alisa Liu, Ximing Lu, Sean Welleck, Peter West, Ronan Le Bras, Yejin Choi, and Hannaneh Hajishirzi. Generated knowledge prompting for commonsense reasoning. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, **Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)**, pp. 3154–3169, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[9] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. **Advances in Neural Information Processing Systems**, Vol. 35, pp. 24824–24837, 2022.

[10] Cheng Li, Jindong Wang, Yixuan Zhang, Kaijie Zhu, Wenxin Hou, Jianxun Lian, Fang Luo, Qiang Yang, and Xing Xie. Large language models understand and can be enhanced by emotional stimuli, 2023.

[11] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers, 2023.

[12] Elvis Saravia. Prompt Engineering Guide. **https://github.com/dair-ai/Prompt-Engineering-Guide**, 12 2022.

[13] 中村健, 乙武北斗, 田辺利文, 吉村賢治ほか. 単一化文法を用いた日本語文の構文解析における並列構造の処理. 第 81 回全国大会講演論文集, Vol. 2019, No. 1, pp. 451–452, 2019.

[14] Taku Kudo. Mecab: Yet another part-of-speech and morphological analyzer. **http://mecab. sourceforge. net/**, 2005.

A　プロンプト例

A.1　ベースライン

以下の文章中に含まれる並列要素の組をすべて漏らさずに列挙してください。並列構造が複数存在したり、入れ子になっているような場合についてもすべて漏らさず列挙してください。与えられた文章中に構造が存在しないと判断した場合には、空の配列のみを返してください。

この時取得した結果は配列 words の要素に追加して、最後には words を含む以下の形式の JSON として表示してください。また、その際に、指示していない値は絶対に出力しないようにしてください。結果は以下の通りですなどの日本語の文言も絶対に出力しないでください

```
# 出力形式
{predict_words : [["",""], ... ]}

# 対象
クエリ: [この部分に対象のクエリを埋め込む]
出力:
```

A.2　具体例

```
# 例 1
クエリ: 私は今日の朝はご飯とみそ汁を食べた
出力:{predict_words: [["ごはん","みそ汁"]]}

# 例 2
クエリ: 学校に登校したが日曜日なので誰もいなかった
出力:{predict_words: []}

# 例 3
クエリ: 日本で食べられる動物には牛、馬などがあり、洋食や和食問わず利用される
出力:{predict_words: [["牛","馬"],["洋食","和食"]]}
```

A.3　原文に含まれていない出力を禁止

この時、元の文に含まれる語を抽出することのみが正しく、元の文に含まれない語を生成することは禁止です。

A.4　並列構造の知識生成

```
# 並列構造の定義
```

並列構造は、言語学や文法において、同じ階層や種類の単語、句、節が文中で連続して並べられることを指します。これは、同等の重要性を持った要素が文中で「並列」されているときに用いられる概念です。特に名詞の並列においては、共通の格で共通の文節に係ることが多いです。

```
# 並列構造の特徴
```

1．並列構造には句を接続する働きを持つ文字列が存在します。この文字列を「並列キー」と呼びます。例文 1 では<<と>>で囲まれた文字列が並列キーです。並列キーとしては「や」や「と」などの並列助詞や、「、」や「,」などの読点、「・」などの中黒などがよく用いられます。
例文 1:『太郎が紅茶に 砂糖 <<と>> ミルク を入 re ru』

2．並列キーの前と後に同じ形式の文要素が存在します。この文要素を「並列要素」と呼びます。例文 2 では{と}で囲まれた文字列が並列要素です。
例文 2:『太郎が紅茶に {砂糖} と {ミルク} を入 re ru』

3．並列要素の単語数は必ずしも同一ではないです。例文における / は単語境界です。例文 3 では、1 つ目の並列要素の単語数は 3 であり、2 つ目の並列要素の単語数は 4 であり、これらの個数は異なっています。
例文 3:『太 郎 が{水/を/飲 m} i,{自 転 車/で/学 校/に/行 k} u』

4．文から並列キーと片方の並列要素を取り除いても意味が通ります。元の文は『太郎が水を飲み, 自転車で学校に行く』ですが、このうち並列要素の片方を除いた例文 4-1,4-2 のいずれも意味が通じています。
例文 4-1:『太郎が {水を飲 m} u』
例文 4-2:『太郎が {自転車で学校に行 k} u』

A.5　元の部分文字列の抽出を強制

取得した文字列に「`」や「「」」などの記号がついていてもそのまま出力してください。

```
# 例 4
クエリ: Java には `if` 文や `for` 文などの文法がある
出力:{predict_words: [["`if` 文","`for` 文"]]}

# 例 5
クエリ: 芥川龍之介の著作には「鼻」や「羅生門」がある
出力:{predict_words: [["「鼻」","「羅生門」"]]}
```

A.6　助詞の補完

元の文は非常に口語的なため必要な助詞が欠落している場合があります。したがって、現在の文面だけから考えるのではなく、必要だと思われる助詞を適切に補完したのちに並列構造を探してください。
ただし、繰り返しになりますが、元の文に含まれる語を抽出することのみが正しいため、補完した助詞は出力に含めないでください。

```
# 例 6
クエリ: 兄と学校行ってきた
出力:{predict_words: []}
```

A.7　感情、役割、深呼吸

困難なタスクですが、あなたは非常に優れているので自信を持ってチャレンジしてみましょう。// 感情
あなたは日本語の母語話者で自然言語処理の専門家です。// 役割
深呼吸をして落ち着いて取り組みましょう。// 深呼吸

A.8　Chain-of-Thought Prompting

思考: 段階的に考えてください。// 0-shot の場合
思考: 並列助詞「と」が含まれているので並列構造が存在します。その前後の「ご飯」と「味噌汁」はどちらも食事に関連し、共通の文節「食べる」に係っているため並列です。// 例 1 の場合
思考: 並列構造の手がかりとなる文節が含まれていないため、並列構造は存在しません。// 例 2 の場合