令和 6 年度茨城大学工学部情報工学科

卒業研究論文

プロンプトに構造化されたデータを付与する効果

所属　情報工学科

著者　深作晃久 (21T4067X)

指導教員　新納浩幸教授

令和 7 年 2 月 4 日 (火)

令和 6 年度茨城大学工学部情報工学科　卒業研究論文

プロンプトに構造化されたデータを付与する効果

著者

深作晃久 (21T4067X)

指導教員

新納浩幸教授

論文要旨

アプリケーションやシステム内で LLM を使用する際には, モデルの出力を構造化されたフォーマットで取得することが必要なことがある.LLM の出力が構造化されている方が, 後段のタスクにおける利用が容易である．特に JSON 形式の出力は簡易で利用しやすい．では LLM の入力に構造化されたデータを使用することは有効なのだろうか．構造化されたデータをプロンプトに加えた場合,LLM の出力は改善されるのか．JSON 形式のデータを LLM でそのまま利用可能であれば, データの処理やシステム間の統合がより簡単になる．本稿では JSON 形式で前段タスクの出力を与えることが, 後段タスクの精度に及ぼす効果について調査する．タスクは条件つきの英日翻訳とし, テストデータは英語版 Wikipedia の本文 10 文書とした．条件は原文の固有名詞を英語のまま訳文に記載することとした. 原文から抽出した固有名詞を原文の言語で JSON 形式のデータとして翻訳の際に付与した．実験の結果, 与えたデータがタスクの精度改善に効果があることが判明した.

目次

第 1 章

序論

1.1 研究の背景

LLM（大規模言語モデル）の活用において, 出力の形式は非常に重要な要素である. 特に近年,LLM で汎用的な言語処理タスクを解くことが広く普及し, これらを組み込んだより複雑なシステムが実用化された. [8] [8] によると,LLM をシステムに組み込んで利用する際の課題の一つに, システムデザインがある. LLM 間や OS といったソフトウェアとの連携である. 特に LLM の入出力の形式は,LLM を一つのコンポーネントとみなすとインターフェイスであるので, システムを構築する際に重要な要素である. LLM が出力する結果の長さや形式にばらつきがある場合, 特に自然言語で出力される場合, 得た結果をそのまま利用することが難しいことがある. 例えば, 自然言語の出力が冗長であったり, 意味のまとまりが不十分であったりする場合, そのままでは目的に即した形で活用することが難しい. こうした場合には, 出力結果を後から加工し, 必要な情報を抽出して再構築する作業が求められることになる. このような加工が手間となり, 効率的に活用できないという問題が生じる. 一方で出力の形式に関して, 構造化されている方が後段タスクにおける利用が容易である. 構造化データは, その特性上, 情報が明確に整理されており, 必要なデータを迅速に取り出すことができるため, 自然言語による出力に比べて活用が容易である. このことは LLM が生成する結果をより目的に沿った形で処理できる可能性が高くなる. 例えば,LLM が JSON や XML のような形式で出力を行うことで, 利用者はそのままプログラム内でデータを扱いやすくなり, 結果の解釈や再利用が効率的に行うことができる.

1.2　研究の目的

LLM の入力に構造化されたデータを使用することは有効なのか, 本研究で調査する.. LLM の入力として構造化されたデータを活用することができれば,LLM をアプリやシステム上で活用する際に有効である．データベースなどにある構造化されたデータを直接 LLM に入力することで, 既存のデータを変換することなく LLM の入力として使用可能になる．本研究では, 特に JSON 形式で構造化されたデータを LLM の入力に加えることが, タスクの精度にどのような影響を与えるのかについて調査を行う．この形式は, データの構造を直感的に理解できるため, タスクごとの最適化に有利であると予想される.

たとえば LLM を用いた分類タスクや推論タスクでは,JSON 形式でデータが整理されているとモデルがその構造を適切に処理し, 正確な判断を下すことができる可能性が高くなる. また JSON は機械可読性が高く,API などでのデータ交換にも適しているため,LLM を活用したシステム開発においても有用な形式といえる．本研究では,JSON 形式の構造化データを LLM の入力に付与することで, タスクの精度が向上するのか調査し, その効果を実証することを目指す．結果として,LLM の活用方法に関する新たな知見を得るとともに, 実際のアプリケーションにおいてどのようにデータ形式を選定すべきかに関する指針を提供することができると期待される.

第 2 章

関連研究

本研究と関連した研究についていくつか紹介する。

2.1 大規模言語モデルの量子化の影響

本研究では実験において大規模言語モデルを量子化して使用している．量子化とはニューラルネットワーク内における計算の桁数を削減することで, 大規模言語モデルのサイズを縮小する技術である．計算速度の向上やメモリ使用量の削減など, 利点があるが欠点も存在する．本研究では翻訳タスクを量子化した LLM に解かせている. その精度に関連する研究として, Kelly Marchisio, Saurabh Dash, Hongyu Chen らによる, 大規模言語モデルの量子化が多言語タスクの精度に与える影響についての研究を紹介する．[6] パラメータ数 103B から 8B の多言語 LLM を対象に、様々な量子化手法を適用し、その影響を調査した．20 以上の言語を対象に自動ベンチマーク、LLM-as-a-Judge 法、人間による評価を使用して詳細な分析を行った．この研究では以下のことが明らかになった．

2.1.1 自動評価と人間による評価の乖離

一つに自動評価と人間による評価結果の乖離がある．量子化によるパフォーマンスの低下が自動指標よりも、人間による評価で大きな影響が見られた．自動評価では量子化の影響が相対的に低い傾向があった．以下の 2.1 に自動評価と人間による評価の差を示す．数字は量子化によって低下したスコアの割合を示す．

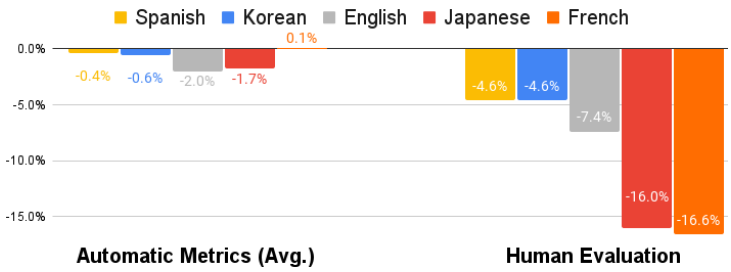

図 2.1: 自動評価と人間による評価の比較 [6] より引用

図 2.1 の結果から, 評価方法によって量子化の影響が変わることが明らかになった. 量子化の影響を正しく把握するには、人間の判断を重視する必要があると主張している.

2.1.2　言語による影響の差異

言語によって量子化の影響が異なり、非ラテン文字の言語が最も大きな影響を受けることが明らかになった. 次の図 2.2 に示す.

		Avg Rel. %Δ			mMMLU			MGSM			Lang. Conf. (Mono)		
		en	Ltn/IE	All	en	Ltn/IE	All	en	Ltn/IE	All	en	Ltn/IE	All
103B	W8	0.1%	-0.3%	-0.3%	0.0%	0.0%	0.0%	0.3%	-0.7%	-1.0%	0.0%	-0.1%	0.0%
	W8A8-sq	-1.2%	-0.6%	-0.7%	-0.1%	-0.4%	-0.5%	-3.4%	-1.3%	-1.6%	0.0%	-0.1%	0.0%
	W8A8	-0.3%	-0.7%	-0.8%	-0.7%	-1.3%	-1.7%	0.0%	-0.9%	-1.0%	-0.1%	0.0%	0.2%
	W4-g	-2.0%	-0.9%	-1.5%	-1.7%	-1.1%	-1.5%	-4.4%	-1.8%	-3.0%	0.0%	0.1%	0.0%
	W4	-3.7%	-3.9%	-4.3%	-3.3%	-3.9%	-4.4%	-7.9%	-8.0%	-8.8%	0.0%	0.1%	0.1%
35B	W8	-0.1%	-0.2%	-0.2%	-0.1%	-0.1%	-0.1%	-0.3%	-0.6%	-0.8%	0.1%	0.1%	0.1%
	W8A8	0.2%	-1.6%	-1.8%	0.0%	0.0%	-0.2%	0.0%	-4.9%	-5.6%	0.7%	0.0%	0.3%
	W4-g	-1.2%	-4.8%	-5.2%	-1.8%	-1.5%	-2.0%	-2.2%	-12.2%	-13.1%	0.4%	-0.6%	-0.4%

図 2.2: 英語と他の言語における量子化コマンドモデルの相対的性能. 英語以外のすべての言語 (All)、英語以外のラテン文字/インド・ヨーロッパ言語 (Ltn/IE)。[6] より引用

図 2.2 によると, ラテン文字の言語が平均 0.7 パーセント性能が低下したのに対し、非ラテン文字の言語は 1.9 パーセント性能が低下した. 本研究で行った英日翻訳はラテン文字の言語から非ラテン文字の言語への翻訳であり, 量子化の影響が比較的大きいと考える.

2.1.3　タスクの難易度と量子化の影響

難易度が高いタスクの方が量子化の影響を受けやすく, 性能の低下が比較的大きくなる. 次の図 2.3 にタスクごとの正解率を示す.

		Avg. Rel. %Δ	mMMLU		MGSM		FLORES En→L2		FLORES L2→En		Language Confusion Monolingual		Language Confusion Cross-lingual	
103B	FP16	-	66.7	-	70.6	-	37.7	-	39.6	-	99.2	-	91.5	-
	W8	-0.2%	66.7	0.0%	69.9	-1.0%	37.7	0.0%	39.6	0.0%	99.2	0.0%	91.2	-0.3%
	W8A8-sq	-0.5%	66.3	-0.5%	69.5	-1.6%	37.8	0.2%	39.1	-1.3%	99.2	0.0%	91.5	0.1%
	W8A8	-0.8%	65.6	-1.7%	69.8	-1.1%	37.7	0.0%	39.1	-1.2%	99.4	0.2%	90.4	-1.2%
	W4-g	-0.9%	65.7	-1.4%	68.6	-2.9%	37.8	0.4%	39.4	-0.5%	99.2	0.0%	90.5	-1.1%
	W4	-2.5%	63.8	-4.3%	64.4	-8.8%	37.1	-1.6%	39.0	-1.6%	99.3	0.1%	92.8	1.4%
35B	FP16	-	59.4	-	49.8	-	32.4	-	35.5	-	98.7	-	66.5	-
	W8	-0.2%	59.3	-0.1%	49.4	-0.7%	32.3	-0.2%	35.4	-0.2%	98.8	0.1%	66.3	-0.2%
	W8A8	0.2%	59.3	-0.2%	47.1	-5.5%	32.9	1.6%	35.8	0.9%	99.0	0.3%	68.9	3.7%
	W4-g	-2.8%	58.2	-2.0%	43.3	-13.1%	31.7	-1.9%	35.3	-0.7%	98.3	-0.4%	67.1	1.0%

図 2.3: 103B および 35B Command モデルの量子化レベルを変化させた場合の、英語以外の言語におけるデータセットごとの平均性能 % Δ は FP16 に対する相対性能 [6] より引用

図 2.3MGSN というタスクでは 35B モデルで性能が平均 13.1% 低下している. タスクの難易度が上がるほど量子化の影響は大きくなり, 出力結果が悪化することを示している.

2.1.4　例外

一方で量子化により出力の精度がほとんど低下しない場合があった. 35B のモデルに W8A8(weight-and-activation quantization at 8-bit) という量子化を行った場合は例外で、翻訳と言語混同の評価で高いパフォーマンスを示したため、全体的にわずかに向上した.

2.1.5　まとめ

LLM が用いる言語によって, 量子化したときの LLM のパフォーマンスに差が生じた. この研究ではモデルの重要な評価基準として、多言語性能を考慮することを主張している.

2.2　大規模言語モデルに構造化された出力を要求することの影響

本研究では LLM の入力に構造化されたデータを使用し, 実験において LLM に構造化された出力を要求している.. 関連する研究として,Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai らによる大規模言語モデルに構造化された出力フォーマットを要求すること

が、モデルのパフォーマンスに与える影響の研究を紹介する. [7] LLM をアプリやシステム上で活用する際に, 出力が構造化されたフォーマット (JSON,XML など) である方が扱いやすい. しかし構造化された出力フォーマットを要求することが、LLM のパフォーマンスに与える影響に関しての研究は少ない. この研究では推論タスクと分類タスクを解くことで調査した. LLM に構造化された出力を要求する方法として以下の 3 つを定義し実験を行った.

1. 出力を JSON フォーマットに制限する方法
2. 出力フォーマットを指定する命令をモデルに与える方法
3. 自然言語で出力を生成した後、それを構造化フォーマットに変換する方法

3 つの方法それぞれで分類タスクと推論タスクを解き,LLM の出力を評価した. 実験の結果以下のことが明らかになった.

- 出力形式の誤りは LLM を解析器として使用することで軽減することができる.
- 推論タスクでは構造化された出力形式を要求することで、LLM の回答の精度が低下する傾向がある. 出力に厳しい制約を課すとほど精度が低下する.
- 分類タスクでは構造化された出力フォーマットを要求することが LLM の精度の向上をもたらす傾向がある.

2.2.1　まとめ

LLM やタスクによって差があるが,LLM に構造化された出力形式を要求すると, パフォーマンスが低下する可能性がある. 出力形式に厳密な制約を課すことが,LLM のパフォーマンスの低下をもたらすことが明らかになった.

2.3　Many-Shot In-Context Learning

本研究の実験で行ったことは, プロンプトエンジニアリングという分野に属する. プロンプトエンジニアリングとは,LLM から目的の出力を得るために LLM への入力であるプロンプトの開発と最適化を行うことである. これに関する執筆時点における新しいアプローチとして,Many-Shot In-Context Learning を紹介する. [1]

LLM に与える例を shot という. 数十個程度であれば few-shot という.

In-Context Learning (ICL) とは、LLM が推論時にプロンプトに添付された具体例から新しいタスクを学習する能力のことである. LLM は例を与えることで, 新しいタスクを LLM の内部データを変更することなく解くことができる. 従来は具体例を数例から多くとも 100 例以下にする,few-shot ICL が主流であった.

しかし,Rishabh Agarwal,Avi Singh,Lei M. Zhang らによる研究で与える具体例を大きく増やす,many-shot ICL が,LLM の出力結果を改善させることが明らかになった. [1] この研究では機械翻訳、要約など複数のタスクにおいて many-shot ICL が few-shot ICL を上回る結果となった. 次の図 2.4 に many-shot ICL と few-shot ICL の比較を示す.

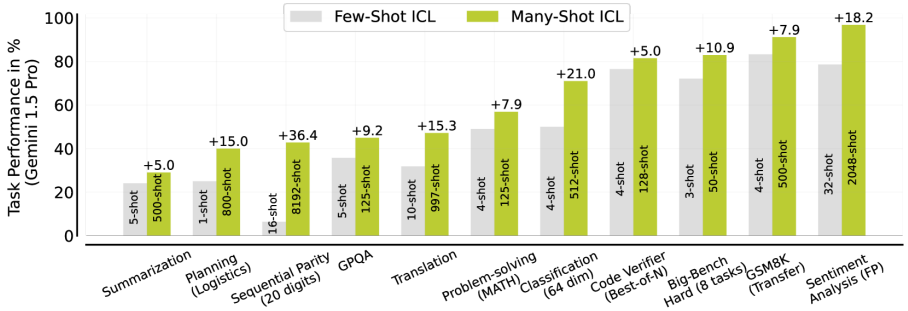

図 2.4: 複数のタスクにおける many-shot ICL と few-shot ICL の比較 [1] より引用

特に難易度の高い非自然言語処理タスクにおいて、many-shot ICL は few-shot ICL を常に上回る結果になった.

さらに多くのタスクで、最適なパフォーマンスを達成するには数百から数千の具体例をプロンプトに与えることが必要であると結論づけている. タスクごとに必要な具体例の数を次の図 2.5 に示す.

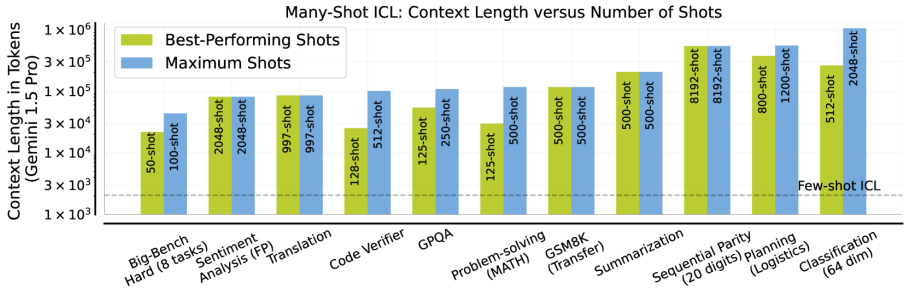

図 2.5: 各タスクでテストされた最良の結果と最大ショット数のコンテキストの長さ. 横の破線は GPT-3 のトークン数である 2048 トークンを表す. [1] より引用

図 2.5 によると最大の性能が得られるのは、shot 数が数十万トークンに達した後であることが多い.

many-shot ICL において与える, 人が作成した数十万のデータの品質を保つことは難しい. [1] この研究ではその課題への対応策として,Reinforced ICL と Unsupervised ICL という 2 つの手法を提案している.

Reinforced ICL

モデルが生成した回答を CL に使用する. 強化学習の手法を用いている.

Unsupervised ICL

Reinforced ICL とは異なり, 問題のみを使用して ICL に使用する.

さらに many-shot ICL の分析を行った. many-shot ICL は LLM がもつバイアスを減じることが示された.LLM は学習に使用するデータから, 偏見を学んでしまう. [9] この偏見は few-shot ICL で取り除くことは難しいとされる. [1]

Many-shot ICL の効果について調査を行った,Yixing Jiang,Jeremy Irvin,Ji Hun Wang,Muhammad Ahmed Chaudhry らによる研究を紹介する. [5] この研究では GPT-4o と Gemini 1.5 Pro を用いて、10 のデータセットで画像分類タスクを対象に実験を行い,many-shot ICL の効果を調査した. 実験の結果,many-shot ICL が few-shot ICL を上回る結果となった.

第 3 章

実験

構造化された出力をプロンプトに加える効果を調査するために, 条件つき英日翻訳タスクにおける結果の比較を行う. 条件は原文の固有名詞を英語のまま訳文に記載することとした. 原文から抽出した固有名詞を原文の言語で JSON 形式のデータとして翻訳の際に付与した. JSON 形式で固有名詞を提供することで,LLM がより正確に固有名詞を処理できるかを調査する.

3.1 使用した事前学習済みモデル

本実験では, 使用した事前学習済みモデルとして,OpenAI 社が開発した GPT-4o と, 東京工業大学および産業技術総合研究所（AIST）によって開発された Llama 3.1 Swallow を採用した. これらのモデルはそれぞれ, 異なる背景と技術に基づいて開発されており, 各モデルが持つ特徴を生かして実験を行った.

まず, GPT-4o は OpenAI 社が提供する最新の大規模言語モデルであり, その強力な自然言語処理能力で広く知られている. GPT-4o は,OpenAI の API を通じて利用することができ, ユーザーはこれを通じて高度な言語生成や理解タスクを実行することができる. 本実験では,OpenAI 社が提供する API を介して GPT-4o を使用した. このモデルは, 多様なテキスト生成, 質問応答, 推論, 要約生成など, さまざまな自然言語処理タスクに対応できる能力を持っており, その性能の高さが実験の実施に大きく寄与した. GPT-4o のモデル数といった詳細な情報は明らかにされていないが, [2] によると本稿執筆時点でトップクラスの性能がある. 次の図 3.1 に GPT-4o と主要な他の LLM の性能を比較を示す.

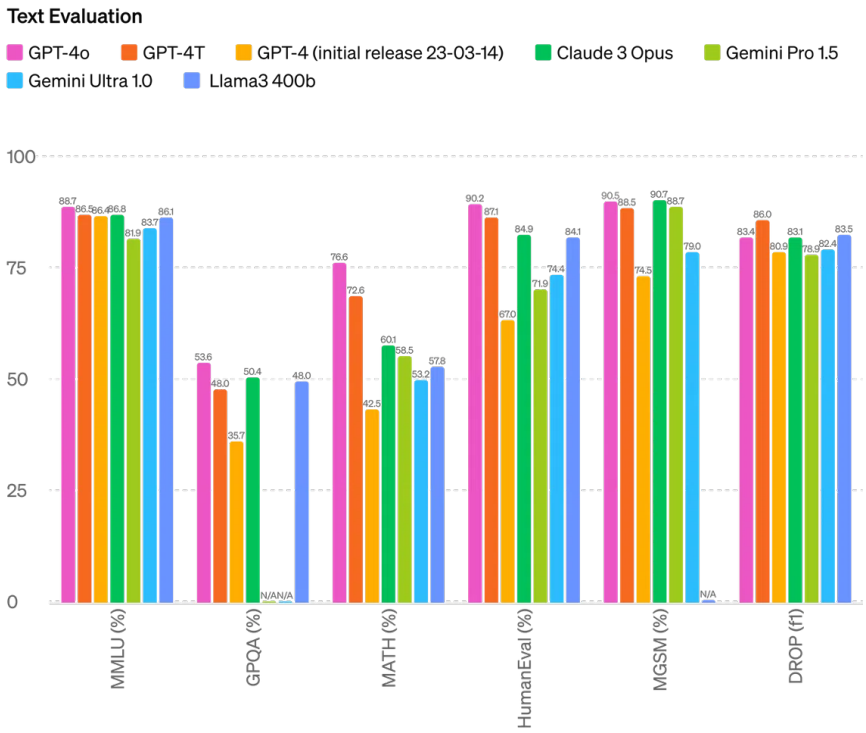

図 3.1: GPT-4o と主要な他の LLM の性能の比較. [2] より引用

図 3.1 によると, 前のモデルである GPT-4 や他のモデルを上回る性能があることが分かる.

次に,Llama 3.1 Swallow は, 東京工業大学および産業技術総合研究所が共同で開発したモデルであり, 特に効率的なパラメータチューニングや大規模なテキストデータセットを用いた事前学習において優れた成果を上げている. Llama 3.1 Swallow は, 特にトランスフォーマーモデルをベースにしたアーキテクチャを採用しており, 英語で事前学習したモデルの, 日本語の対応能力を上げたオープンソースの大規模言語モデルである. Meta Llama 3.1 モデルを継続的な事前学習を行うことで, 日本語を扱う能力を向上させている. [4] [4] によると, あらかじめ英語コーパスをもとに事前学習したモデルを使用することで, 一から日本語コーパスをもとにモデルを学習させるよりも, 少ないコストでモデルの学習を行うことができる. このモデルは公開されているモデルの中では, トップクラスの日本語を扱う性能がある. そのため本実験においてはこのモデルを使用することにした. 本実験では,HuggingFace というプラットフォームで公開されている Llama 3.1 Swallow のバージョン「Llama-3.1-Swallow-8B-Instruct-v0.3」を使用した.

web サイト,`huggingface.co/tokyotech-llm/Llama-3.1-Swallow-8B-Instruct-v0.3` で公開されている.Python ライブラリの Huggingface から利用可能だ. [3] による

と,Llama-3.1-Swallow-8B-Instruct-v0.3 大規模な日本語 Web コーパス（Swallow Corpus Version 2）, 日本語と英語の Wikipedia 記事, 数学とコーディングのコンテンツなどからサンプリングされた約 2000 億トークンを継続的な事前学習に使用している. このバージョンは特に指示に基づくタスクに強い性能を発揮するようにチューニングされている. 本実験で採用した理由は, 英語から日本語への翻訳に適していると考えるためと,GPT-4o と比べて小型の LLM における効果を調査するためである.

また, 実験におけるハードウェアの制約により,Llama 3.1 Swallow を使用する際にはモデルのパラメータを 4 ビットに量子化して運用した. 量子化とはモデルのサイズを縮小し, 計算資源の消費を抑える方法である. 量子化を行うとモデルの性能は低下する傾向にある. [6] これにより限られたハードウェアリソースの中で, できるだけ効率的に実験を進めることができた.

3.2　使用したデータセット

データセットには Wikipedia データセットを使用した. このデータセットは世界中の多くの言語で利用可能な web 百科事典である Wikipedia を基にして制作されており, その内容は非常に多岐にわたる分野をカバーしている. Wikipedia から品質の良い文章を大量に取得することができる. そのため本実験においてはこのデータセットを使用することにした.

Huggingface というプラットフォームで公開されている Wikipedia データセットを利用した. このプラットフォームは様々な機械学習モデルやデータセットを公開しており, ユーザーが容易にアクセスできる形で提供されている. Wikipedia データセットも Huggingface 上で公開されており,'graelo/wikipedia' という名前で利用可能である. web サイト,`huggingface.co/datasets/graelo/wikipedia` で公開されている.Python ライブラリの Huggingface から利用可能だ. このようなアクセスのしやすさは実験において便利であり, 迅速にデータセットを取得して利用できる点で有益である. 本実験では,Wikipedia データセットに含まれる 6705754 件の英語の文書からランダムに 10 件の文書を抽出して使用した. このようにして選ばれた文書は,Wikipedia 内で扱われている様々な分野にわたるトピックが含まれており, 実験の多様性を確保するために適切であると考えた. ただし, プロンプトのトークン数に制限があるため, 各文書の冒頭から最初の 10 文を抽出して使用することにした. これはプロンプトが長すぎると処理速度や計算資

源に大きな影響を及ぼす可能性があるため, トークン数を効率的に抑えるための対策である. 10 文という制限を設けることで, 効率良く実験を行うことができると判断した. このようにして選ばれた文書は, 実験における処理や解析に使用した.

3.3　実験手法

本実験では英日翻訳タスクを次の 2 つに分けて解くアプローチを採用した.

1. 原文から固有名詞を抽出し, それを JSON のデータにするタスク
2. 翻訳を行うタスク

固有名詞の有無に関して 2 通りの翻訳方法を採用した。一つは,JSON データを事前に作成し, それを翻訳過程に組み込んで翻訳を行う方法. もう一つは JSON データを使用せずにそのまま翻訳を行う方法である. JSON 形式で構造化された固有名詞のデータを追加することが, 翻訳結果に与える影響を調査することができる. 1 と 2 のタスクを GPT-4o,Llama-3.1-Swallow-8B-Instruct-v0.3 でそれぞれ行った.

3.3.1　原文から固有名詞の表を得る

1. 原文から固有名詞を抽出し、それを JSON 形式でデータ化するタスクの解く手法を説明する.

モデル名:GPT-4o の場合 Open AI 社が提供する Python ライブラリ,Langchain の機能を用いて JSON 形式の固有名詞のデータを取得した. 以下の図 3.1 に英文から固有名詞のデータを取得するソースコードを示す.

ソースコード 3.1: 固有名詞の抽出に用いたプロンプト

```
1     class Entity(TypedDict):
2         "Task: Extract proper nouns such as place names and personal
              names,organization names,brand names,era names,event names
              from the sentence."
3         proper_noun: Annotated[str,...,"Extracted proper nouns"]
4         type: Annotated[str,...,"Types of extracted proper nouns"]
5     class NounList(TypedDict):
6         entities: Annotated[List[Entity],...,"List of entities found
              in the text"]
7
```

```
8     structured_llm=llm.with_structured_output(NounList)
```

”Task: Extract proper nouns such as place names...”がタスクの指示である. 英語で固有名詞を抽出することを指示している. 固有名詞の具体例も記述した.

モデル名:Llama-3.1-Swallow-8B-Instruct-v0.3 の場合プロンプトに自然言語で英文から固有名詞を抽出し,JSON 形式で出力することを指示した. 以下の 3.2 に使用したプロンプトテンプレートを示す. PROMPT_DICT でプロンプトのひな型を作成し,create_prompt 関数でひな型を元にプロンプトを作成する.

{instruction},{input}の部分を埋めて LLM への入力とした.

ソースコード 3.2: 固有名詞の抽出に用いたプロンプト

```
1     PROMPT_DICT = {
2     "prompt_input": (
3         "以下に、あるタスクを説明する指示があり、それに付随する入力が更
      なる文脈を提供しています。"
4         "リクエストを適切に完了するための回答を記述してください。\n\n"
5         "### 指示:\n{instruction}\n\n### 入力:\n{input}\n\n### 応答:"
6
7     ),
8     "prompt_no_input": (
9         "以下に、あるタスクを説明する指示があります。"
10        "リクエストを適切に完了するための回答を記述してください。\n\n"
11        "### 指示:\n{instruction}\n\n### 応答:"
12    ),
13    }
14    def create_prompt(instruction, input=None):
15        """
16       Generates a prompt based on the given instruction and an
      optional input.
17       If input is provided, it uses the 'prompt_input' template from
      PROMPT_DICT.
18       If no input is provided, it uses the 'prompt_no_input' template
      .
19
20       Args:
21           instruction (str): The instruction describing the task.
22           input (str, optional): Additional input providing context
      for the task. Default is None.
23
```

```
24          Returns:
25              str: The generated prompt.
26          """
27           if input:
28                # Use the 'prompt_input' template when additional input is
        provided
29                return PROMPT_DICT["prompt_input"].format(instruction=
        instruction, input=input)
30           else:
31                # Use the 'prompt_no_input' template when no additional
        input is provided
32                return PROMPT_DICT["prompt_no_input"].format(instruction=
        instruction)
```

以下の 3.3 に LLM への指示を示す.

ソースコード 3.3: LLM への指示

```
1     Extract proper nouns such as place names and personal names,
      organization names,brand names,era names,event names from the
      sentence.
2     出力は形式 JSON({'entities': [{'proper_noun': 'Faith Rogers', 'type
      ': 'personal name'}, {'proper_noun': 'Heavener Runestone Park', '
      type': 'place ' name}]})に
      する。
```

英語で固有名詞を抽出し,JSON 形式による出力を要求している.JSON 形式の出力の例を与えている.

3.3.2　翻訳を行うタスク

使用したプロンプトを以下に示す. 構造化されたデータを与えない場合, 下線部分は削除して使用した.

タスク: 次の文章を日本語に翻訳する。固有名詞（地名、人名、時代名など）は英語のまま記載する。ただし条件に従う。翻訳の例：Jon が America を建国した。

条件：表（固有名詞とその説明がペアになったデータ）を参考にする。

表：America： country... 文章:America is...

上記のプロンプトを使用して,GPT-4o,Llama-3.1-Swallow-8B-Instruct-v0.3 それぞれにおいて, 翻訳を構造化された固有名詞のデータの有無,2 通り行った.

3.4　実験結果

構造化された出力をプロンプトに加えた方が, 望ましい出力となった. 以下にモデルごとの実験結果を示す. 評価は人手で行い, 評価基準は翻訳文に原文中の固有名詞を英語のまま記載できた割合とした. 原文と翻訳文で対応する文どうしの比較を行った. 以下の表 3.1 に実験結果を示す. 表 3.1 の網羅率の定義は 翻訳文中の固有名詞の総数/原文中の固有名詞の総数とする.

表 3.1: 実験結果

モデル名	網羅率	
	JSON データあり	JSON データなし
GPT-4o	0.84	0.75
Llama-3.1-Swallow-8B-Instruct-v0.3	0.43	0.20

2 つのモデル両方で JSON データをプロンプトに加えることにより, 精度の向上が確認できた.

次の 3.2 に各文書に対応する翻訳文の評価結果を示す.

表 3.2: 各文書ごとの網羅率

モデル	JSON データの有無	文書番号									
		1	2	3	4	5	6	7	8	9	10
gpt-4o	JSON データなし	0.52	0.64	0.96	1.00	0.63	0.92	0.86	0.64	1.0	0.76
	JSON データあり	0.86	0.88	0.92	0.84	0.78	0.92	1.0	0.64	1.0	0.95
Swallow-8B-Instruct-v0.3	JSON データなし	0.13	0.00	0.15	0.00	0.00	0.00	0.86	0.20	0.25	0.43
	JSON データあり	0.52	0.60	0.80	0.00	0.56	0.19	0.09	0.40	0.25	0.86

表 3.2 より, 網羅率に大きな差が生じたのは Llama-3.1-Swallow-8B-Instruct-v0.3 の方であった. 指示にまったく従っていない翻訳文が減少した. 個々の文書における結果を比較しても,GPT-4o の方が良い結果となった.

次に LLM が原文から作成した,JSON 形式の固有名詞データについて調査する. 評価は人手で行い, 評価基準は原文中の固有名詞を抽出できた割合とした. 翻訳文とは異なり, 原文との対応を考慮することはできなかった. 表 3.3 の網羅率の定義は JSON データ中

の固有名詞の総数/原文中の固有名詞の総数とする.

表 3.3: 原文から生成した JSON 形式の固有名詞データ. 原文と対応するデータの比較ではない.

モデル名	網羅率
GPT-4o	0.60
Llama-3.1-Swallow-8B-Instruct-v0.3	0.47

JSON データの網羅率を見ると, 翻訳文の網羅率と比較して,Llama-3.1-Swallow-8B-Instruct-v0.3 では JSON データを与えた場合と近い結果になったが, GPT-4o では大きく低下している. GPT-4o でパフォーマンスが低下した原因は,1 文ごとに固有名詞を抽出することを指示していないため, 重複を避けたのではないかと推察する.

次の表 3.4 に各文書ごとの比較結果を示す. 原文と対応するデータの比較ではない.

表 3.4: JSON データの各文書ごとの網羅率. 原文と対応するデータの比較ではない.

モデル	文書番号									
	1	2	3	4	5	6	7	8	9	10
gpt-4o	0.83	0.60	0.50	0.31	0.45	0.58	0.91	0.52	0.88	0.76
Swallow-8B-Instruct-v0.3	0.56	0.44	0.46	0.69	0.33	0.42	0.91	0.24	0.13	0.38

与える JSON データの網羅率と翻訳文における網羅率の間に関係があるのか調査する.

次の図 3.2 と 3.3 に JSON データと翻訳文の網羅率の散布図を示す.

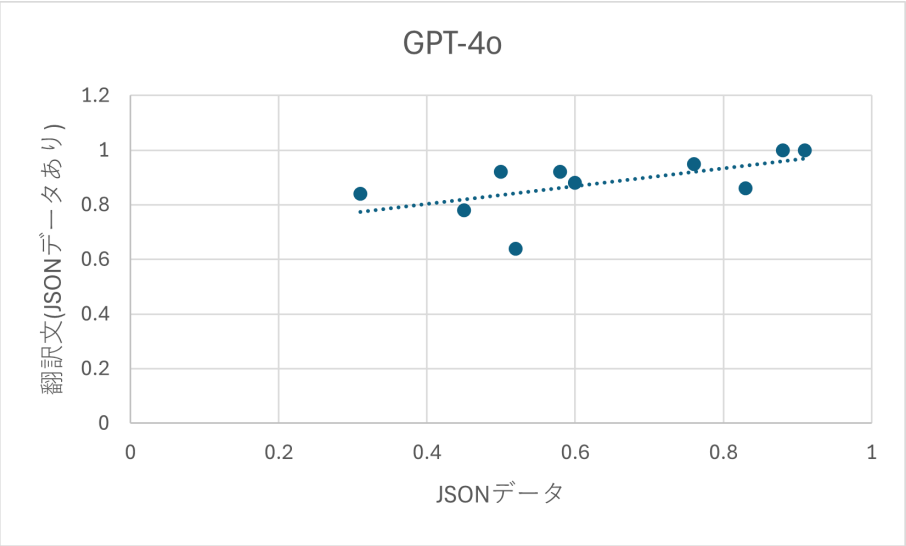

図 3.2: JSON データと翻訳文の比較

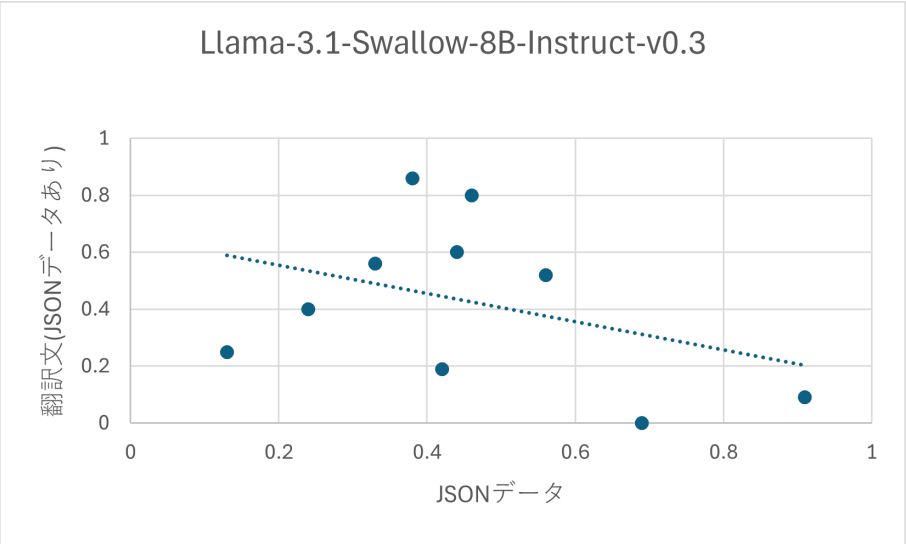

図 3.3: JSON データと翻訳文の比較

上記の比較について分析を行う．3.2 における相関係数は 0.602 であった．3.3 における相関係数は-0.38 であった．GPT-4o の場合,JSON データの網羅率が高いほど翻訳文における網羅率も高くなったが,Llama-3.1-Swallow-8B-Instruct-v0.3 では異なる結果になった．Llama-3.1-Swallow-8B-Instruct-v0.3 では外れ値の影響が考えられるが, データ数が少ないため断定することはできない．

第 4 章

考察

実験結果を基に考察を行う.GPT-4o,Llama-3.1-Swallow-8B-Instruct-v0.3, 両方で JSON 形式のデータがタスクの精度向上に有効であった. 図図 3.1 によると,Llama-3.1-Swallow-8B-Instruct-v0.3 の方が効果が大きく, 精度が約 2 倍になっている. 一般的な個人向けのコンピューター内のみで実行可能な, 小型の LLM における効果の方が大きいと考える. 結論付けるにはさらに多くの同様の小型の LLM で実験を行う必要があるが, 実験結果はこのことを示唆していると考える.

JSON 形式のデータが中により多くの固有名詞が含まれていれば, 翻訳文の結果が改善すると予想した. 図 3.2 と図 3.3 によると, そうではないと考える. 図 3.1 より, 使用する LLM の性能に結果が左右されると結論付ける.

図 3.1 より, 本稿の実験で使用した, 訳文中に原文の固有名詞を原文の言語で記載するタスクにおいて, JSON 形式のデータを付与することは, 効果的であると考える.

本稿ではタスクの需要や意義についての調査は行わなかったが, このタスクを LLM で解くことは実際の翻訳において役に立つ可能性がある. 特に専門用語が多く含まれる文献の翻訳において, 有益だと予想する.

第 5 章

結論

JSON 形式で構造化されたデータを LLM のプロンプトに加えることが, 条件つき翻訳タスクの精度の向上に有効であると結論づける.

第 6 章

課題

いくつかの課題が浮き彫りになった. これらの課題を考慮した追実験を行うことで, 構造化されたデータをプロンプトに加える効果のさらなる検証が可能となる.

6.1 タスク

本稿では翻訳タスクのみ行った. 推論など他のタスクにおける効果について, 調査を行う必要がある.

6.2 データ数

今回の実験では 10 文書を使用した. さらに数を増やしても本稿と同様の結果を得ることが可能であるか, 調査する必要があると考える. 本稿で使用したデータセットには, 英語だけで約 600 万件に及ぶ文書が存在する. 複数言語による実験が可能であれば, 十分な数のデータを用意することができると考える.

6.3 評価方法

本稿では人手で行ったが, 手間がかかるためより効率良く行う手法が必要である. 人手で行う信頼性を維持しつつ, 効率よく翻訳文と原文の比較を行う手法があれば大規模なデータを使用した実験の評価が可能だと考える. 本稿では自動評価を検討したが取りやめた. 人手による評価と比較して, 固有名詞を LLM などにより正確に認識し抽出することと, 原文と翻訳文の対応を考慮して評価することの精度と実装に疑問が生じたためであ

る. 以上の課題を解決することが, 自動評価を実装するために必要であると考える.

6.4　量子化の影響の調査

本研究における量子化の影響について, 比較実験を行うべきだと考える. 参考文献 [6] で示された通り, 本実験においても影響があると考える. しかし, ハードウェア上の制約により困難であった. 比較実験を行うことができる LLM を使用するか, 実行環境の整備を行うことで実験が可能になると考える.

6.5　使用する LLM

本稿の実験の図 3.1 では 8B の小型 LLM の方が, 出力が改善された. 構造化されたデータをプロンプトに加える効果は小規模なモデルの方が大きい可能性がある. さらに複数のモデルで実験を行いこのことに関して調査する必要があると考える.

謝辞

本研究は国立国語研究所の共同研究プロジェクト「テキスト読み上げのための読みの曖昧性の分類と読み推定タスクのデータセットの構築」及び JSPS 科研費 23K11212 の助成を受けています．本研究を進めるにあたり多くのご指導を頂いた指導教員の新納浩幸教授に感謝申し上げます．また同研究室の皆さまには基礎学習において多大なご助言とご支援を賜り、誠にありがとうございました。

参考文献

[1] Rishabh Agarwal, Avi Singh, Lei M. Zhang, Bernd Bohnet, Luis Rosias, Stephanie Chan, Biao Zhang, Ankesh Anand, Zaheer Abbas, Azade Nova, John D. Co-Reyes, Eric Chu, Feryal Behbahani, Aleksandra Faust, and Hugo Larochelle. Many-shot in-context learning, 2024.

[2] Open AI. Hello gpt-4o. `https://openai.com/index/hello-gpt-4o/`, 2024. Accessed on January 30, 2025.

[3] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, and Angela Fan et al. The llama 3 herd of models, 2024.

[4] Kazuki Fujii, Taishi Nakamura, Mengsay Loem, Hiroki Iida, Masanari Ohi, Kakeru Hattori, Hirai Shota, Sakae Mizuki, Rio Yokota, and Naoaki Okazaki. Continual pre-training for cross-lingual llm adaptation: Enhancing japanese language capabilities, 2024.

[5] Yixing Jiang, Jeremy Irvin, Ji Hun Wang, Muhammad Ahmed Chaudhry, Jonathan H. Chen, and Andrew Y. Ng. Many-shot in-context learning in multimodal foundation models, 2024.

[6] Kelly Marchisio, Saurabh Dash, Hongyu Chen, Dennis Aumiller, Ahmet Üstün, Sara Hooker, and Sebastian Ruder. How does quantization affect multilingual llms?, 2024.

[7] Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung yi Lee, and Yun-Nung Chen. Let me speak freely? a study on the impact of format restrictions on performance of large language models, 2024.

[8] Matei Zaharia, Omar Khattab, Lingjiao Chen, Jared Quincy Davis, Heather

Miller, Chris Potts, James Zou, Michael Carbin, Jonathan Frankle, Naveen Rao, and Ali Ghodsi. The shift from models to compound ai systems. `https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/`, 2024.

[9] 白藤 大幹, 竹中 誠, 斉藤 辰彦, and 木村 泰知. Task arithmetic に基づく言語モデルにおけるバイアス低減手法の検討. **人工知能学会第二種研究会資料**, 2024(AGI-028):06, 2024.

付録

使用したプログラムの一部を以下に添付する.

ソースコード .1: 固有名詞の抽出と翻訳を GPT-4o で行う

```
import getpass
from typing import Optional,List
from typing_extensions import Annotated, TypedDict

from langchain_openai import ChatOpenAI

llm = ChatOpenAI(model="gpt-4o",temperature=1.0)
#######
from typing_extensions import Annotated, TypedDict,Optional


# 抽出
#文章から地名、人名といった固有名詞を取り出す。""
class Entity(TypedDict):
    "Task: Extract proper nouns such as place names and personal names,
    organization names,brand names,era names,event names from the
    sentence."
    proper_noun: Annotated[str,...,"Extracted proper nouns"]
    type: Annotated[str,...,"Types of extracted proper nouns"]

class NounList(TypedDict):
  entities: Annotated[List[Entity],...,"List of entities found in the
    text"]

structured_llm=llm.with_structured_output(NounList)
# And a query intented to prompt a language model to populate the data
    structure.
from datasets import load_dataset
```

```
# 文章から文取り出す 10
def extract_sentences(data):
    import re
    data=re.split(r'[.!?]\s*', data)
    data=[s.strip() for s in data if s]
    data=data[:10]
    text=""
    for s in data:
        text=text+s
    return str(text)


graelo_wiki = load_dataset("graelo/wikipedia","20230901.en",
    trust_remote_code=True)
train_data=graelo_wiki['train']
list_of_text_data_num=[2169922, 1295447, 1122070, 4734751,
    3832340, 3335368, 2768600, 3337392, 4171473, 2165501,
    3695548, 5973486, 5122182, 4633228, 4643937, 1018851,
    1221556, 4324300, 6463232, 1674464, 2875838, 6524156,
    574344, 4320674, 4288884, 4628522, 2559364, 251880, 5636235,
     279501]
result="" # 翻訳した結果
result_list=[] # 翻訳した結果をに保存 pkl
result_of_json_data="" # 取り出した json
result_list_of_json_data=[] # 取り出したをに保存 jsonpkl
for i in list_of_text_data_num:
    data=train_data['text'][i]
    text=extract_sentences(data[:3000])
    table=structured_llm.invoke(text)
    query="""タスク次の文章を日本語に翻訳する。固有名詞地名、人名、時代名
    など:()は英語のまま記載する。ただし条件に従う。翻訳の例：がを建国し
    た。条件：表 JonAmerica固有名詞とその説明がペアになったデータ ()を参考
    にする。表："""
    query=query+str(table)
    query=query+"文章："+text
    #print(query)
    output=llm.invoke(query)
    result=result+str(i)+'\n'
    result=result+str(output.content)+'\n'
    result_list.append(output.content)
    # を文字列に json
    result_of_json_data=result_of_json_data+str(i)+'\n'
```

```
58      result_of_json_data=result_of_json_data+str(table)+'\n'
59      result_list_of_json_data.append(str(table))
60  print(result_list)
61  # 固有名詞の表を与える ver
62  # ファイルを開く 書き込みモード ( 'w')
63  with open('/home/', 'w') as file:
64      file.write(result)
65  import pickle
66  # ファイルに保存
67  with open('/home/', 'wb') as f:
68      pickle.dump(result_list, f)
69  # 抽出したを保存 json
70  # ファイルを開く 書き込みモード ( 'w')
71  with open('/home/', 'w') as file:
72      file.write(result_of_json_data)
73  import pickle
74  # ファイルに保存
75  with open('/home/', 'wb') as f:
76      pickle.dump(result_list_of_json_data, f)
```

ソースコード .2: 固有名詞の抽出を swallow で行う

```
1       import os
2       import sys
3       import getpass
4       from typing import Optional,List
5       from typing_extensions import Annotated, TypedDict
6       import torch
7       from torch import cuda,bfloat16
8       from transformers import AutoTokenizer,AutoModelForCausalLM
9       import transformers
10      from typing_extensions import Annotated, TypedDict,Optional
11
12
13      model_id = "tokyotech-llm/Llama-3.1-Swallow-8B-Instruct-v0.3"
14      token = ''
15      quant_config = transformers.BitsAndBytesConfig(
16          load_in_4bit=True,
17          bnb_4bit_quant_type='nf4',
18          bnb_4bit_use_double_quant=True,
19          bnb_4bit_compute_dtype=bfloat16
```

```
20      )
21      model = transformers.AutoModelForCausalLM.from_pretrained(
22          model_id,
23          trust_remote_code=True,
24          token=token,
25          quantization_config=quant_config,
26          torch_dtype=torch.bfloat16,
27          low_cpu_mem_usage=True,
28          device_map="auto",
29      )
30      tokenizer = AutoTokenizer.from_pretrained(
31          model_id,
32          token=token
33      )
34      PROMPT_DICT = {
35          "prompt_input": (
36              "以下に、あるタスクを説明する指示があり、それに付随する入力
        が更なる文脈を提供しています。"
37              "リクエストを適切に完了するための回答を記述してください。
        \n\n"
38              "### 指示:\n{instruction}\n\n### 入力:\n{input}\n\n### 応
        答:"
39
40          ),
41          "prompt_no_input": (
42              "以下に、あるタスクを説明する指示があります。"
43              "リクエストを適切に完了するための回答を記述してください。
        \n\n"
44              "### 指示:\n{instruction}\n\n### 応答:"
45          ),
46      }
47
48      def create_prompt(instruction, input=None):
49          """
50         Generates a prompt based on the given instruction and an
        optional input.
51         If input is provided, it uses the 'prompt_input' template from
        PROMPT_DICT.
52         If no input is provided, it uses the 'prompt_no_input' template
        .
53
54         Args:
55             instruction (str): The instruction describing the task.
```

```
56          input (str, optional): Additional input providing context
    for the task. Default is None.
57
58      Returns:
59          str: The generated prompt.
60      """
61       if input:
62            # Use the 'prompt_input' template when additional input is
     provided
63           return PROMPT_DICT["prompt_input"].format(instruction=
    instruction, input=input)
64       else:
65            # Use the 'prompt_no_input' template when no additional
    input is provided
66           return PROMPT_DICT["prompt_no_input"].format(instruction=
    instruction)
67  ##############
68  instruction_example = """Extract proper nouns such as place names
    and personal names,organization names,brand names,era names,event
    names from the sentence出力は.形式
    JSON({'entities': [{'proper_noun': 'Faith Rogers', 'type': '
    personal name'}, {'proper_noun': 'Heavener Runestone Park', 'type':
     'place '
    name}]}
69  )にする。"""
70  input_example = "On a sunny afternoon in Paris, Emma Watson and
    Leonardo DiCaprio met at the Louvre Museum to discuss their
    upcoming project, 'The Great Gatsby', with director Christopher
    Nolan, while musician Taylor Swift and actor Robert Downey Jr.
    enjoyed coffee, discussing the latest trends in fashion from
    designers like Gucci, Prada, and Louis Vuitton."
71  prompt = create_prompt(instruction_example, input_example)
72
73  input_ids = tokenizer.encode(
74      prompt,
75      add_special_tokens=False,
76      return_tensors="pt"
77  )
78
79  tokens = model.generate(
80      input_ids.to(device=model.device),
81      max_new_tokens=700,
```

```
        temperature=0.99,
        top_p=0.95,
        do_sample=True,
    )

    output = tokenizer.decode(tokens[0], skip_special_tokens=True)
    print(output)
    "次の文章から応答の:形式の部分を取り出す。取り出したのみ出力する。JSONJSON"
    from langchain_openai import ChatOpenAI
    llm = ChatOpenAI(model="gpt-4o",temperature=1.0)
    query="次の文章から応答の:形式の部分を取り出す。取り出したのみ出力する。文章 JSONJSON:"+output
    output=llm.invoke(query)
    print(output.content)
```

ソースコード .3: JSON データを付与した翻訳を swallow で行う

```
    import os
    import sys

    from langchain_openai import AzureChatOpenAI
    import getpass
    from typing import Optional,List
    from typing_extensions import Annotated, TypedDict
    import torch
    from torch import cuda,bfloat16
    from transformers import AutoTokenizer,AutoModelForCausalLM
    import transformers
    import pickle
    ###############
    model_id = "tokyotech-llm/Llama-3.1-Swallow-8B-Instruct-v0.3"
    token = '' トークン huggingface
    quant_config = transformers.BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_quant_type='nf4',
        bnb_4bit_use_double_quant=True,
        bnb_4bit_compute_dtype=bfloat16
    )
    model = transformers.AutoModelForCausalLM.from_pretrained(
        model_id,
        trust_remote_code=True,
```

```
        token=token,
        quantization_config=quant_config,
        torch_dtype=torch.bfloat16,
        low_cpu_mem_usage=True,
        device_map="auto",
    )
    tokenizer = AutoTokenizer.from_pretrained(
        model_id,
        token=token
    )
    PROMPT_DICT = {
        "prompt_input": (
            "以下に、あるタスクを説明する指示があり、それに付随する入力が更なる文脈を提供しています。"
            "リクエストを適切に完了するための回答を記述してください。\n\n"
            "### 指示:\n{instruction}\n\n### 入力:\n{input}\n\n### 応答:"

        ),
        "prompt_no_input": (
            "以下に、あるタスクを説明する指示があります。"
            "リクエストを適切に完了するための回答を記述してください。\n\n"
            "### 指示:\n{instruction}\n\n### 応答:"
        ),
    }

    def create_prompt(instruction, input=None):
        """
       Generates a prompt based on the given instruction and an optional input.
       If input is provided, it uses the 'prompt_input' template from PROMPT_DICT.
       If no input is provided, it uses the 'prompt_no_input' template.

       Args:
           instruction (str): The instruction describing the task.
           input (str, optional): Additional input providing context for the task. Default is None.

       Returns:
```

```
60              str: The generated prompt.
61          """
62          if input:
63              # Use the 'prompt_input' template when additional input is
 provided
64              return PROMPT_DICT["prompt_input"].format(instruction=
instruction, input=input)
65          else:
66              # Use the 'prompt_no_input' template when no additional
input is provided
67              return PROMPT_DICT["prompt_no_input"].format(instruction=
instruction)
68
69
70      from typing_extensions import Annotated, TypedDict,Optional
71
72      from langchain_openai import ChatOpenAI
73
74      query="""次のの出力から、LLM応答に対応する部分を取り出して。取り出し
        た部分のみを出力すること###"""
75
76
77      # And a query intented to prompt a language model to populate the
        data structure.
78      from datasets import load_dataset
79      #graelo_wiki = load_dataset("graelo/wikipedia","20230901.en",
        trust_remote_code=True)
80      # 文章から文取り出す 10
81      def extract_sentences(data):
82          import re
83          data=re.split(r'[.!?]\s*', data)
84          data=[s.strip() for s in data if s]
85          data=data[:10]
86          text=""
87          for s in data:
88              text=text+s
89          return str(text)
90      with open('/home/', 'rb') as f:
91          list_of_json_data = pickle.load(f)
92      graelo_wiki = load_dataset("graelo/wikipedia","20230901.en",
        trust_remote_code=True)
```

```
train_data=graelo_wiki['train']
list_of_text_data_num=[2169922, 1295447, 1122070, 4734751,
3832340, 3335368, 2768600, 3337392, 4171473, 2165501,
3695548, 5973486, 5122182, 4633228, 4643937, 1018851,
1221556, 4324300, 6463232, 1674464, 2875838, 6524156,
574344, 4320674, 4288884, 4628522, 2559364, 251880, 5636235,
 279501]
result=""
result_list=[]
result_of_json_data="" # 取り出した json
result_list_of_json_data=[]
for i,json_data in zip(list_of_text_data_num,list_of_json_data):
    data=train_data['text'][i]
    text=extract_sentences(data[:3000])
    # 翻訳
    table=json_data
    instruction_example = "次の文章を日本語に翻訳する。固有名詞地
名、人名、時代名など()は英語のまま記載する。ただし条件に従う。翻訳の
例：がを建国した。条件：表JonAmerica固有名詞とその説明がペアになった
データ()を参考にする。"
    input_example = "表:"+str(table)+"文章:"+text
    prompt = create_prompt(instruction_example, input_example)
    input_ids = tokenizer.encode(
        prompt,
        add_special_tokens=False,
        return_tensors="pt"
    )
    tokens = model.generate(
    input_ids.to(device=model.device),
    max_new_tokens=1000,
    temperature=0.99,
    top_p=0.95,
    do_sample=True,
    )
    output = tokenizer.decode(tokens[0], skip_special_tokens=True
)
    print(output)
    query="""次のの出力から、LLM応答に対応する部分を取り出して。取り
出した部分のみを出力すること###"""
    query+=output
    output=llm.invoke(query)
    output=output.content
```

```
125         #print(output)
126         result=result+str(i)+'\n'
127         result=result+str(output)+'\n'
128         print(i)
129         #print(result)
130         result_list.append(output)
131
132     # ファイルを開く 書き込みモード( 'w')
133     with open('/home/', 'w') as file:
134         file.write(result)
135     import pickle
136     # ファイルに保存
137     with open('/home/', 'wb') as f:
138         pickle.dump(result_list, f)
139     sys.exit()
140     # 抽出したを保存 json
141     # ファイルを開く 書き込みモード( 'w')
142     with open('/home/', 'w') as file:
143         file.write(result_of_json_data)
144     import pickle
145     # ファイルに保存
146     with open('/home', 'wb') as f:
147         pickle.dump(result_list_of_json_data, f)
```