令和 7 年 2 月 26 日

卒業論文

gemini-1.5-flash を用いたトリプル生成のファインチューニングとその評価

東京都立大学

システムデザイン学部電子情報システム工学科

相馬研究室

指導教員：相馬 隆郎 教授

下田真生

学修番号：21141097

第1章　序論

1．1　　はじめに

近年 ChatGPT や Gemini をはじめとする大規模言語モデル(LLM：Large Language Models)の利用が急増している。一般的な質問の回答生成をはじめ、教育機関で用いられたり、専門的な研究データに対して用いられたりと、その活躍は多岐にわたる。

元来 LLM とは大量のテキストデータと高度なディープラーニングを用いて構築されたものであり、従来の言語モデルと比較して計算量・データ量・パラメータ数という 3 つの要素を大幅に強化することでより高度な言語理解を実現している。これは NEC による LLM の解説[1]でも説明されている。さらに最近ではこのような進化により、画像や音声、動画の生成も可能な AI(LLM も含め生成 AI と呼ばれている)も利用されている。機械が人間の言語を理解し、さらに機械自ら様々なコンテンツを生成することが可能になっているのである。

しかし、そんな万能に思える生成 AI にもいくつか課題点が残されている。それは出力が誤っていたり、情報元が不明確であったり、再現性が担保されなかったりすることである。出力の誤りについて、出力全体が完全に間違っている場合や一部が間違っている場合など、その形態は様々である。このような課題はどれだけ元データを増やしたり入力過程を改善したりしても、完全に正確性を保証することは難しいと考えられ、長年問題視されている。

そこで本研究では、後述するナレッジグラフ推論チャレンジという企画における、LLM を用いたトリプル生成のタスクを元に、ファインチューニング(以下 1.2 で説明)を用いて生成 AI の正確性の向上を目指す。

第 1 章では先述したナレッジグラフ推論チャレンジの概要と先行研究について、第 2 章では本研究の研究概要と方法について、第 3 章では研究結果とその評価・考察について述べていく。

1．2　ファインチューニング

本来生成 AI は一般的で幅広いタスクに対応しているため、一般知識としてインプットされていない情報やデータインプット以後の内容については、入力に対して正しい回答が出力できなかったりそもそも回答の生成ができなかったりする。そこで用いるのがファインチューニングであり、追加したい情報に関して、入力とそれに対応する出力をセットでベースモデルとなる AI に学習させることで、未知の情報に対しても正しい出力を得られるようにする手法ある。図にすると以下の図 1 のような流れである。

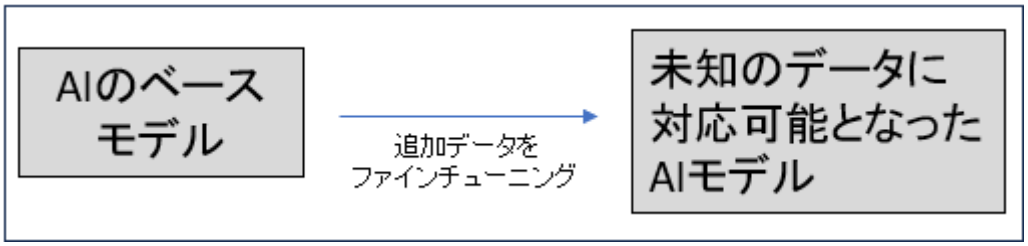

図 1：ファインチューニングの図解

本研究で使用したベースモデルは Gemini が提供する gemini-1.5-flash であり、ベースモデルの他に学習回数・一度にモデルに与えるデータ数・誤差をどの程度反映させるかのパラメータなどを設定することができる。実際に用いたコード例を以下図 2 に示す。

```
operation = genai.create_tuned_model(
    #学習モデル・データ・ID の指定
    source_model=base_model.name,
    training_data=training_data,
    id=name,
    #学習の回数
    epoch_count=30,
    #一度にモデルに与えるデータ数
    batch_size=32,
    #誤差をどの程度反映するかのパラメータ
    learning_rate=0.001,
)
```

図 2：ファインチューニングの設定例

1．3　ナレッジグラフ推論チャレンジ

ナレッジグラフ推論チャレンジ[2]とは、LLM を用いたナレッジグラフの構築というタスクを課題として設定した、生成 AI を用いて推理小説(シャーロック・ホームズシリーズ)の結末を予測するという企画である。具体的には、小説を短文に分けそれをトリプルに分割、トリプルの集合体であるナレッジグラフを構築し、その知識の集合体を用いて AI で結末の推論というものである。ここでいうトリプルとは 1 文を主語(subject)・述語(hasPredicate など)・目的語(what など)に分解したもののことを言い、トリプルと課題解決の流れを図示したものは以下の図 3, 図 4 のようになる。尚、本研究は課題解決のうち短文をトリプルに分けるタスクを扱った。

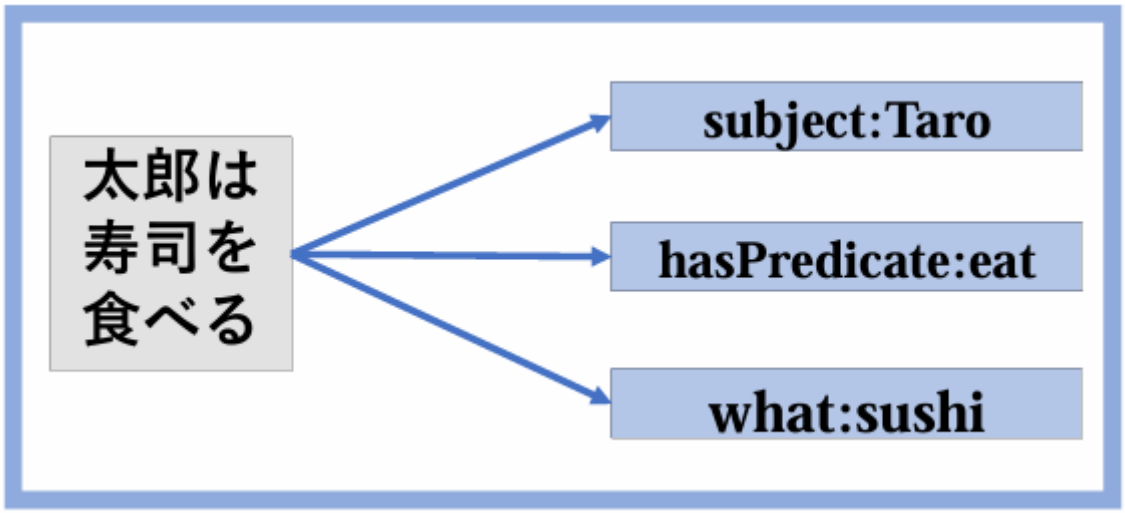

図 3：：トリプルの一例

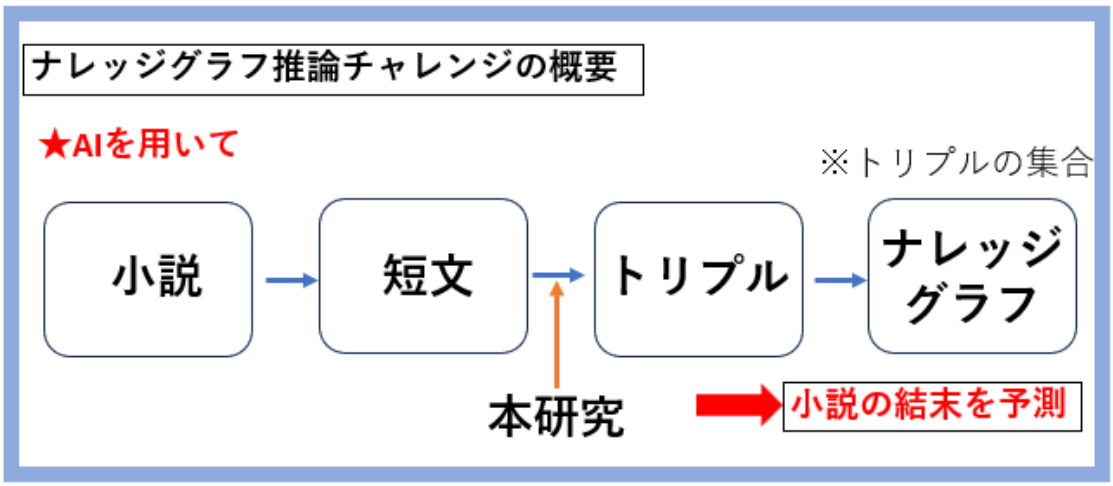

図 4：ナレッジグラフ推論チャレンジの概要

各ステップの中でも、本研究で扱っているトリプル生成とそれを用いてのナレッジグラフの構築についてさらに具体的に説明していく。小説の各シーンとトリプルの組み合わせは公式で提示されており、以下の図 5, 図 6 のように確認できる。

```
PREFIX kgc: <http://kgc.knowledge-graph.jp/ontology/kgc.owl#>
SELECT DISTINCT *
FROM <http://kgc.knowledge-graph.jp/data/SpeckledBand>
#FROM <http://kgc.knowledge-graph.jp/data/DancingMen>
#FROM <http://kgc.knowledge-graph.jp/data/ACaseOfIdentity>
#FROM <http://kgc.knowledge-graph.jp/data/DevilsFoot>
#FROM <http://kgc.knowledge-graph.jp/data/CrookedMan>
#FROM <http://kgc.knowledge-graph.jp/data/AbbeyGrange>
#FROM <http://kgc.knowledge-graph.jp/data/SilverBlaze>
#FROM <http://kgc.knowledge-graph.jp/data/ResidentPatient>
WHERE {
  <http://kgc.knowledge-graph.jp/data/SpeckledBand/1> ?p ?o .
}
```

図 5：小説の各シーン(この場合は「まだらのひも」シーン 1)とトリプルの組み合わせを出力するために用いる SPARQL クエリ画面([3]を引用)

kgc:source	ヘレンがホームズの家にやってくる
kgc:source	Helen comes to Holmes' house
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	kgc:Situation
kgc:hasPredicate	http://kgc.knowledge-graph.jp/data/predicate/come
kgc:subject	http://kgc.knowledge-graph.jp/data/SpeckledBand/Helen
kgc:to	http://kgc.knowledge-graph.jp/data/SpeckledBand/house_of_Holmes

図 6：図 5 を実行した場合の出力画面([3]を引用)

ナレッジグラフ推論チャレンジでは最終的にナレッジグラフの構築を目標としているため、各シーンの ID を主語として、シーンに付随する情報をまとめている。つまり各シーンの ID を大元として、そこからそのシーンの各トリプルや元となる文、関連する他のシーン ID へと関係をつなぎ、ナレッジグラフとするのである。そうすることでシーン間の関係もつなぐことができ、膨大な量の情報をナレッジグラフとして構築できるのである。図示をすると以下の図 7 のようになる。

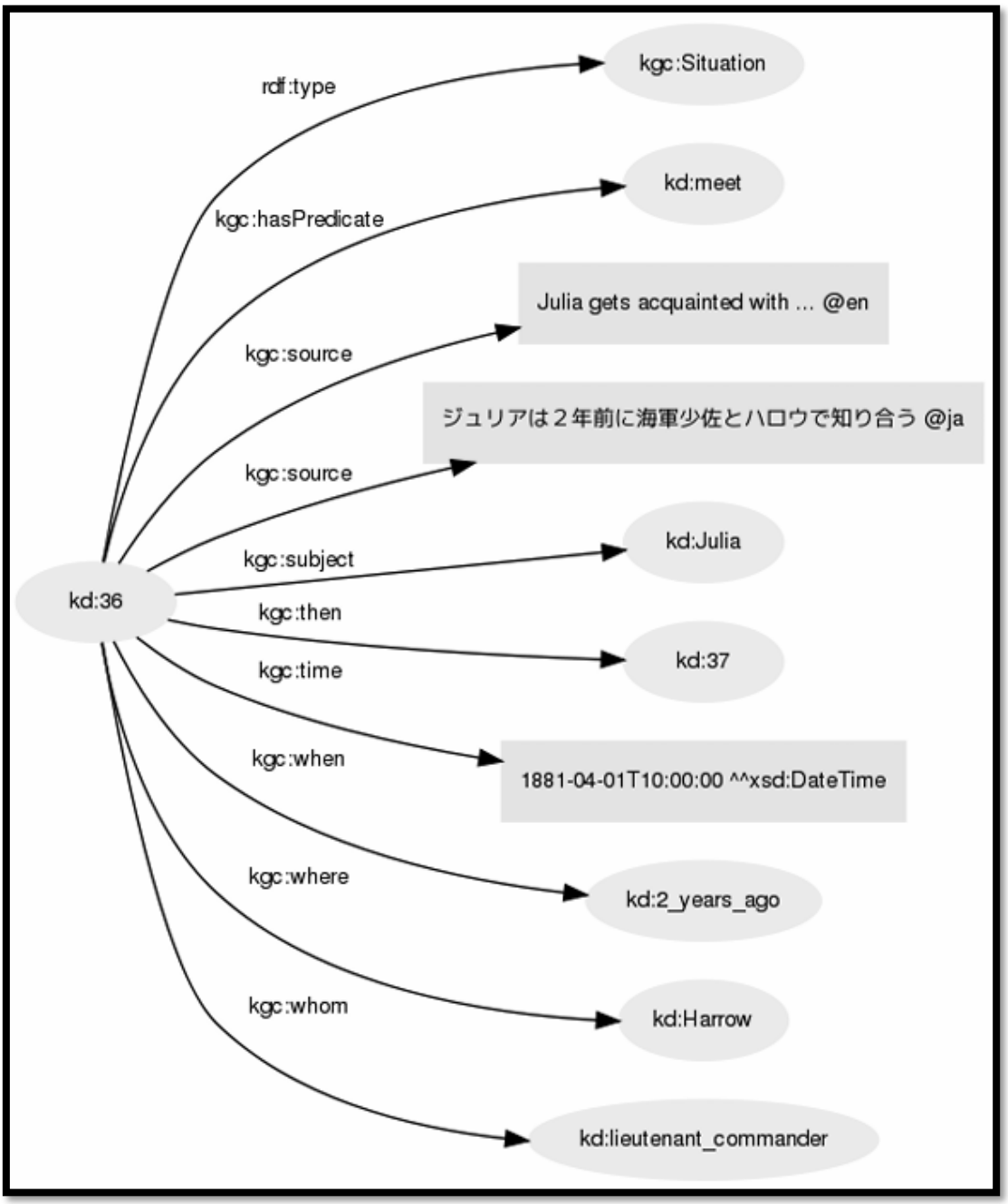

図 7：シーン ID36 におけるトリプル等をつないだ例([4]から引用)

このようにして用意された多くの情報源によって構築されたナレッジグラフは以下の図8ようになる。本研究はトリプル生成までを目的としたため、公式が一例として出しているナレッジグラフを提示する。ここでも SPARQL クエリを用いている。また実際のナレッジグラフは膨大であり、画面に入りきらない(もしくは入れようとすると細かすぎて内容が視認できない)ため図8では一部を抜粋して表示し、次の図9で1シーンに注目したナレッジグラフを提示する。

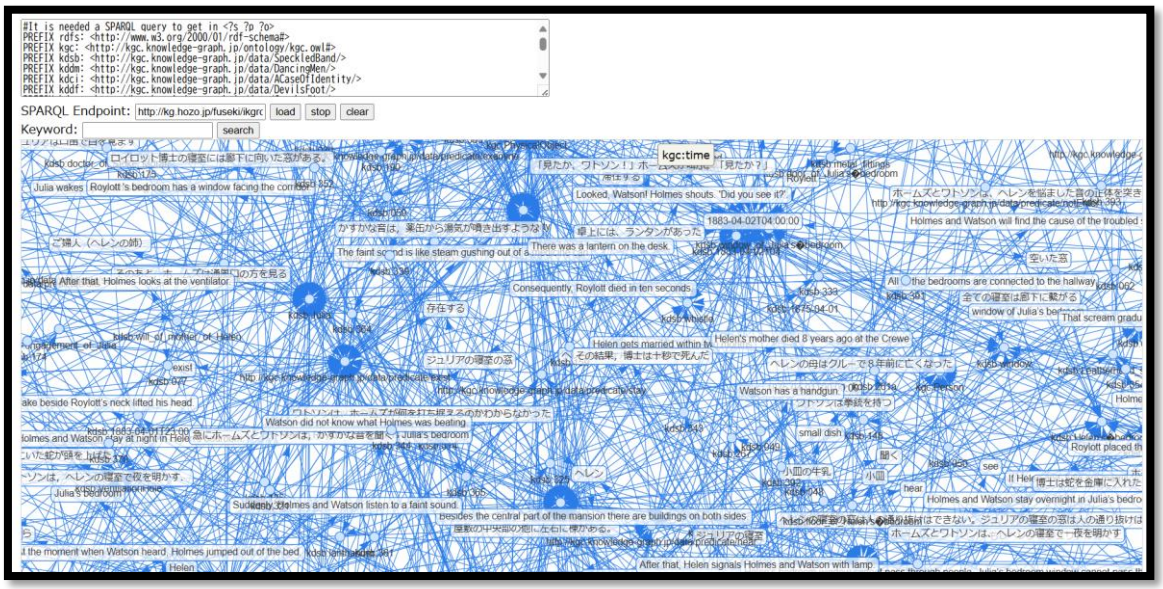

図 8：ナレッジグラフの一部([5]より引用)

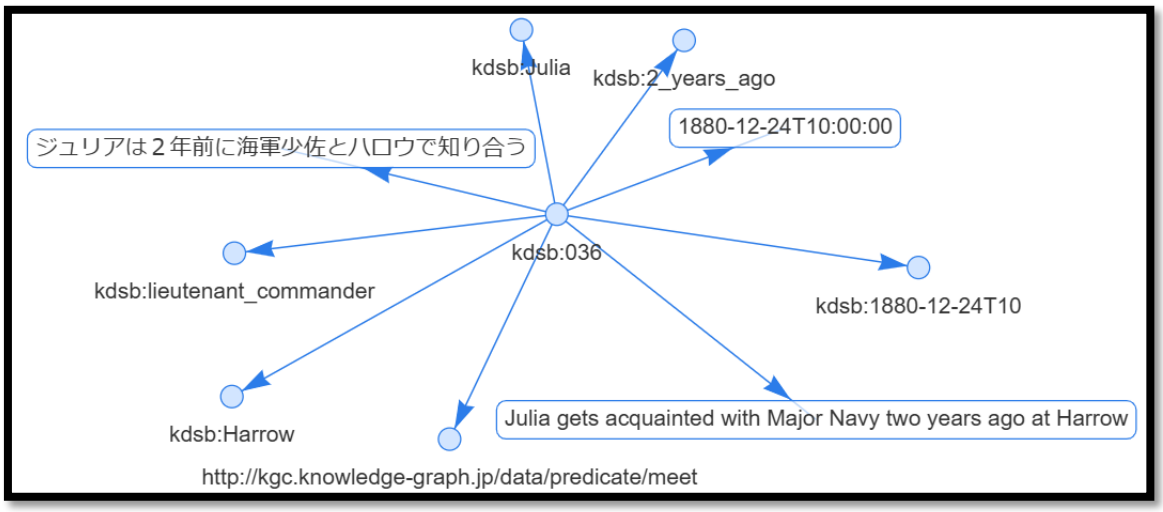

図 9：シーン ID36 におけるナレッジグラフ([5]より引用)

図9の集合体が図8のようになり、膨大な量の情報を含んだナレッジグラフが構築されるのである。

1．4　先行研究

富士通による先行研究(以下先行研究と表記)[6]では、本研究と同じように各シーンからのトリプル生成のタスクに取り組んでいる。先行研究ではトリプル生成を最終タスクとし、シーン間の関係を表すトリプルは消去、シーン ID は scene_x で統一としている。

まず学習データの作成として、先述した各シーンとトリプルの組み合わせを用いて、入力を各シーンのソース文、出力を各シーンのトリプルとしたデータセットを用意している。先述したシーン間の関係を表すトリプルの消去とシーン ID の統一を含めた加工前と加工後の入力と出力は以下の図 10，図 11 のように提示されている。

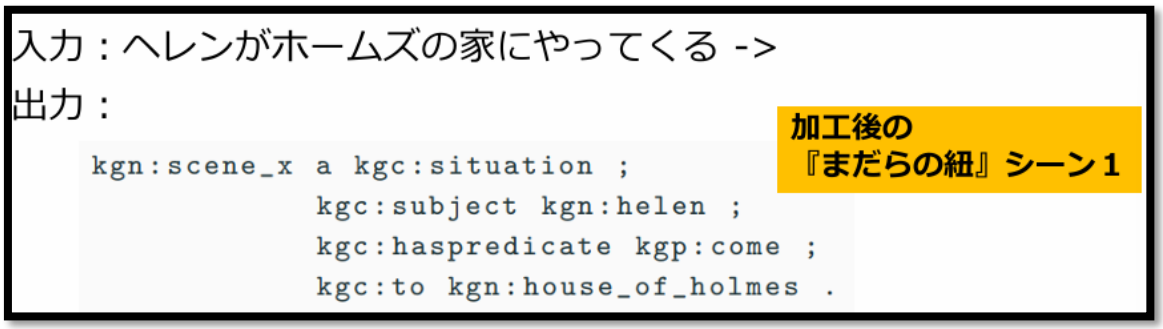

図 10：加工後のデータセット(この場合は小説「まだらのひも」を使用)([6]より引用)

図 11：加工前のデータセット(この場合は小説「まだらのひも」を使用)([6]より引用)
※prompt：入力，completion：出力

ナレッジグラフ推論チャレンジ公式からは「まだらのひも」を含め 8 つの小説に関する各シーンとトリプルが公開されており、先行研究では学習データとして「まだらのひも」以外の 7 つの小説(2632 シーン)を使用している。また、ファインチューニングでは入力文を日本語文・英語文とする 2 種類のデータセットを用意し、それぞれを 3epochs 学習したモデルと日本語文のデータセットを 1epochs 学習したモデルを用意している。

次に上記のデータを用いたファインチューニングを行ったモデルの評価について、学習データには使用しなかった「まだらのひも」(396 シーン)を用いて出力の正確性を確かめている。評価基準は以下の 4 つを用いている([6]より引用)。また、各評価の具体例として次の図 12～図 14 を提示している。

- Loadability：生成されたものが元データと同形式のトリプルとして認識可能な割合
- Precision：生成されたものが正解のシーンにも存在する割合
- Recall：正解のシーンにあるものが，生成されている割合
- F1：Precision, Recall の調和平均

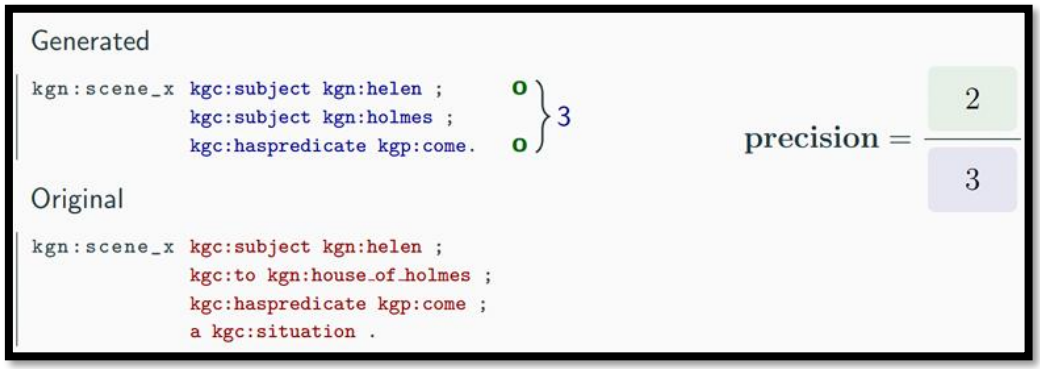

図 12：Precision の評価例([6]より引用)

図 13：Recall の評価例([6]より引用)

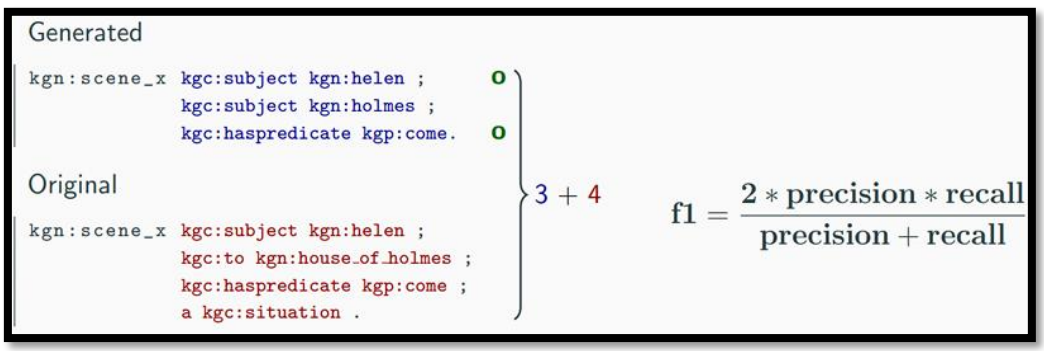

図 14：F1 の評価例([6]より引用)

上記の評価基準を用いた評価は以下の図 15 ように提示されている。尚、使用したモデルは先述した 3 種類のモデルである。併せて、先述した図 7 に示すような全トリプル一つ一つに対する性能評価も行っており、それは以下の図 16 のように提示されている。

	F1	Precision	Recall	Loadbility
ja 3 epochs	0.356	0.410	0.325	1.00
en 3 epochs	0.405	0.471	0.367	1.00
ja 1 epoch	0.336	0.388	0.305	0.997

図 15：各モデルの評価([6]より引用)

各predicateを含むシーン数
#p:生成データ，#t:正解データ

predicate	f1	precision	recall	#p	#t
rdf:type	0.717	0.717	0.717	396.000	396.000
kgc:subject	0.591	0.593	0.588	393.000	396.000
kgc:haspredicate	0.538	0.533	0.544	323.000	316.000
kgc:hasproperty	0.276	0.292	0.263	72.000	80.000
kgc:whom	0.238	0.333	0.185	15.000	27.000
kgc:what	0.257	0.242	0.274	178.000	157.000
kgc:when	0.010	0.029	0.006	70.000	318.000
kgc:time	0.000	0.000	0.000	59.000	295.000
kgc:where	0.171	0.192	0.154	73.000	91.000
kgc:from	0.157	0.200	0.129	20.000	31.000
kgc:to	0.150	0.188	0.125	32.000	48.000
kgc:how	0.175	0.250	0.135	20.000	37.000
kgc:infosource	0.016	0.013	0.020	77.000	49.000

図 16：トリプルの種類ごとの評価([6]より引用)

この結果に対し富士通はまず図 15 について、生成されたトリプルのうちほとんどが元データと同形式で読み込み可能であること、日本語より英語を入力文としたモデルの方が性能が良いこと、そもそも全体的に性能が良いとは言えないこと、といった内容を分析結果としている。次に図 16 について、出現頻度が高く各トリプルに該当する情報が入力文から読み取りやすいもの(図 16 内の黄色枠を指す)は性能評価が高いこと、学習データと評価用データそれぞれが含むトリプルの種類ごとの量の違いによってそもそもトリプルによっては生成頻度が少ないものがあること(図 16 内の赤枠を指す)、といった内容を分析結果としている。

これらの内容から先行研究では、元のデータとの完全一致を正解として評価した場合、先述の評価基準において 1 に近いような性能の良いモデルはなかなか作れないこと、元とするデータごとにトリプルの種類や各トリプルの出現頻度などが異なるため、学習データと評価用データの組み合わせによっては一部のトリプルの出力性能が低くなる可能性があることが読み取れる。

1．5　研究目的

ここまでの内容をふまえ、本研究では同義語の許容とデータセットの種類の単純化で、性能評価の高いモデルを作成すること、学習回数等ファインチューニング設定の調整により少ないデータ数で正確な出力を得ること、以上の 2 点に焦点を当ててトリプル生成のモデルを作成した。先述した評価基準の評価を可能な限り高くすることを最終的な目的とする。

第2章　研究内容

2．1　　概要

本研究では第 1 章 1.4 で記した先行研究の内容に基づき、ナレッジグラフ推論チャレンジの企画タスクの一つをもとに、小説の 1 シーンからトリプルを生成する AI モデルをファインチューニングを用いて作成した。尚、先行研究同様シーン間の関係を考慮しないものとする。学習データの作成、ファインチューニング、学習済みモデルの評価、結果からファインチューニング設定の調整、再評価、といった流れで精度の高い AI モデルの作成を目指した。各工程について以下で述べていく。

2．2　　学習データの作成

第 1 章 1.3 で示したように、ナレッジグラフ推論チャレンジ公式から小説の 1 シーンとトリプルの組み合わせが提供されている。本研究では 8 つの小説のうち「まだらのひも」における組み合わせを学習データとした。「まだらのひも」には約 400 シーンあるが、ファインチューニングにかかる時間の都合上本研究では 200 シーンを用いた。改めて入力と出力の組み合わせの例を以下の図 17 に示す。記法は先行研究に基づく。

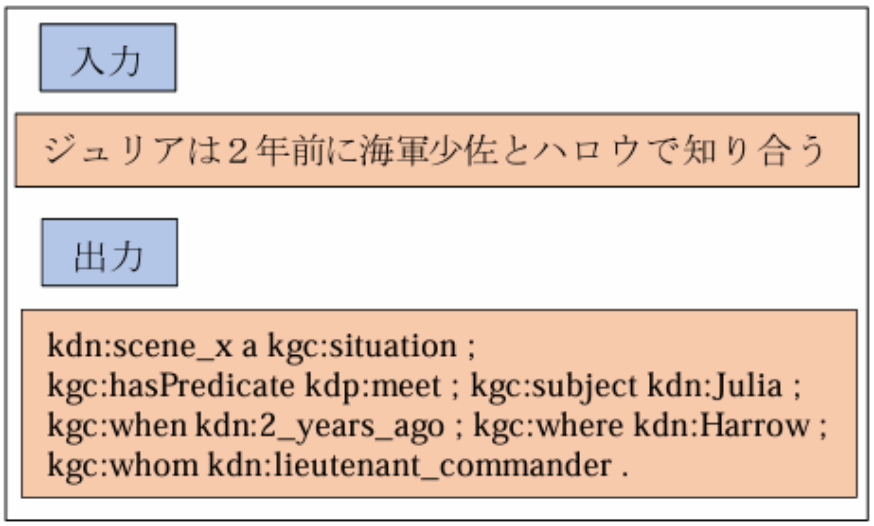

図　17：学習データの組み合わせ例

ここで公式による 1 シーンとトリプルの組み合わせについて、一部の組み合わせでは小説のシーン状況を加味してソース文を意訳したトリプルやソース文から読み取れないトリプルを表記しているものがある。シーン間の関係を考慮しないことに基づき、あくまでソース文のみから読み取れる範囲のトリプルを出力とするため、次の図 18 のように修正を行った。

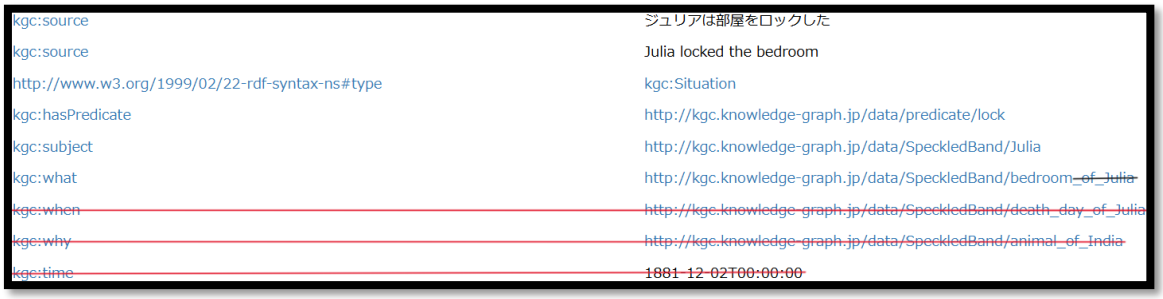

図 18：学習データの修正(黒線：意訳部分の削除，赤線：シーン間のつながりの削除)([3]より引用)

このような修正を加えることで各ソース文から読み取れるシンプルなトリプルを AI が学習し、実際にその AI モデルを使った推論を行う際に誤った出力が出る可能性を低くすることができると考えられる。

また、トリプルの種類を少数化することでも同様の効果が期待できる。そのため、例えば屋敷の反対という場所を示す内容が文に含まれていた場合、目的語として「kgc:opposite kdn:mansion」という組み合わせを公式は提示しているが、「kgc:where kgc:opposite_mansion」という組み合わせに修正することでトリプルの種類の少数化を図った。他にも場所を示す目的語でいうと kgc:near, kgc:right, kgc:left などを公式は使用していたため、それらにも修正を加え、最終的に以下のようなトリプルのみを用いて学習データを作成した。

・文の種類を示すトリプル

Situation(状況を示す文), Statement(人物の主張を示す文), Thought(人物の想像や思考を示す文), Talk(人物から人物への発言を示す文)

・主語を表すトリプル

subject(主語)

・述語を表すトリプル

hasPredicate(基本的な動詞), hasProperty(状態・性質※主に形容詞)

・目的語を表すトリプル(to, from に関しては出現頻度が高かったので修正せずに使用)

what(何を・に), whom(誰を・に), where(場所), when(時間), how(副詞), to(～へ), from(～から), infoSource(Statement や Thought の文において誰に焦点を当てた文なのかを示す)

※修飾語などもトリプルでは目的語とみなす

2．3　ファインチューニング

本研究では第 1 章 1.2 で述べたように Gemini が提供する gemini-1.5-flash をベースモデルとして使用した。用意した 200 セットの学習データを用いてファインチューニングを行ったが、第 1 章 1.2 で示したファインチューニング設定のうち学習回数(epoch_count)を調整して、複数のモデルを作成した。初期段階では 5 回前後のモデルを作成したが正しい出力が得られず、本研究の評価対象として使用したものは学習回数 20 回, 30 回, 50 回のモデルとなった。尚、学習回数 50 回のモデルに関しても正しい出力は得られなかったが、それに関しての評価分析は第 3 章に記す。学習データのファイル読み込みからファインチューニング実行までのコードを以下の図 19～図 21 に示す。

```
import json

#学習データのファイル(json 形式)から読み込んだデータを training_data に代入
with open("madara_test_kai_200.json", "r") as f:
    training_data = json.load(f)
```

図 19：学習データの読み込みコード

```
#Google の生成 AI モデルにアクセス
!pip install -q -U google-generativeai

#Gemini の利用
from google.colab import userdata
import google.generativeai as genai

#API キーの読み込み
GEMINI_API_KEY = userdata.get("GEMINI_API_KEY")
genai.configure(api_key=GEMINI_API_KEY)

#ファインチューニングするベースモデルを設定
import google.generativeai as genai
base_model = genai.get_base_model('models/gemini-1.5-flash-001-
tuning')
```

図 20：Gemini の呼び出しとベースモデルの設定コード

```
#training_data を使ったファインチューニング
import random
import os
from google.cloud import storage

#モデル名の指定
name = f'generate-num-{random.randint(0,10000)}'

#create_tuned_model を呼び出す
operation = genai.create_tuned_model(
    #学習モデル・データ・ID の指定
    source_model=base_model.name,
    training_data=training_data,
    id=name,
    #学習の回数
    epoch_count=30,
    #一度にモデルに与えるデータ数
    batch_size=32,
    #誤差をどの程度反映するかのパラメータ
    learning_rate=0.001,
)
```

図 21：ファインチューニングの実行コード

2．4　評価方法

まずファインチューニングの評価として学習回数と平均損失(mean_loss)の関係を表すグラフを用いる。平均損失とははモデルの予測と実際の値との間の差を平均化したもので、0に近づけば近づくほど十分な学習ができていることを表す。グラフを得るためには以下の図 22 のようなコードを用いる。

```
#ファインチューニングの評価
import pandas as pd
import seaborn as sns

#評価するモデルの指定（この場合は直前に作成されたモデル）
model = operation.result()
snapshots = pd.DataFrame(model.tuning_task.snapshots)
#（epoch:学習回数, mean_loss:平均損失）
sns.lineplot(data=snapshots, x = "epoch", y="mean_loss")
```

図 22：ファインチューニングの評価グラフを得るコード

後述する第 3 章の結果で記すような学習回数と平均損失の関係を示すようなグラフが得られるため、それを元にファインチューニング設定の再調整を行うことができ、精度の高い AI モデルへと修正していくことができる。

次にモデルの性能評価として、第 1 章 1.4 で示した先行研究と同様の評価基準を用いる。この評価基準について、例えば「A(場所)に椅子がある」という文の場合、「chair exist A」と「A have chair」という 2 種類の表記方法があり、本研究では同義であればどの表記方法でも正解として扱った。それに伴い、ソース文の解釈を複数通りに増やし、その中で最も高い評価を得られるものをソース文として扱った(上記の例ならば「chair exist A」と「A have chair」をどちらもソース文候補とみなし、生成されたトリプルと同じ表記の方をそのシーンのソース文として扱うということ)。実際にナレッジグラフを構築する場合も、最終的には各人物やの行動や物の状態がどうであるか、そのつながりがどのようになるのかが重要であるため、公式が出している組み合わせのみを正解とせず性能評価を行うことでより実践的な結果を得られると考える。

第3章　研究結果・考察

３．１　ファインチューニングの結果

ファインチューニングの結果として、第２章 2.4 でも述べたように学習回数と平均損失の関係を表すグラフを作成した。以下の図 23 に示す。

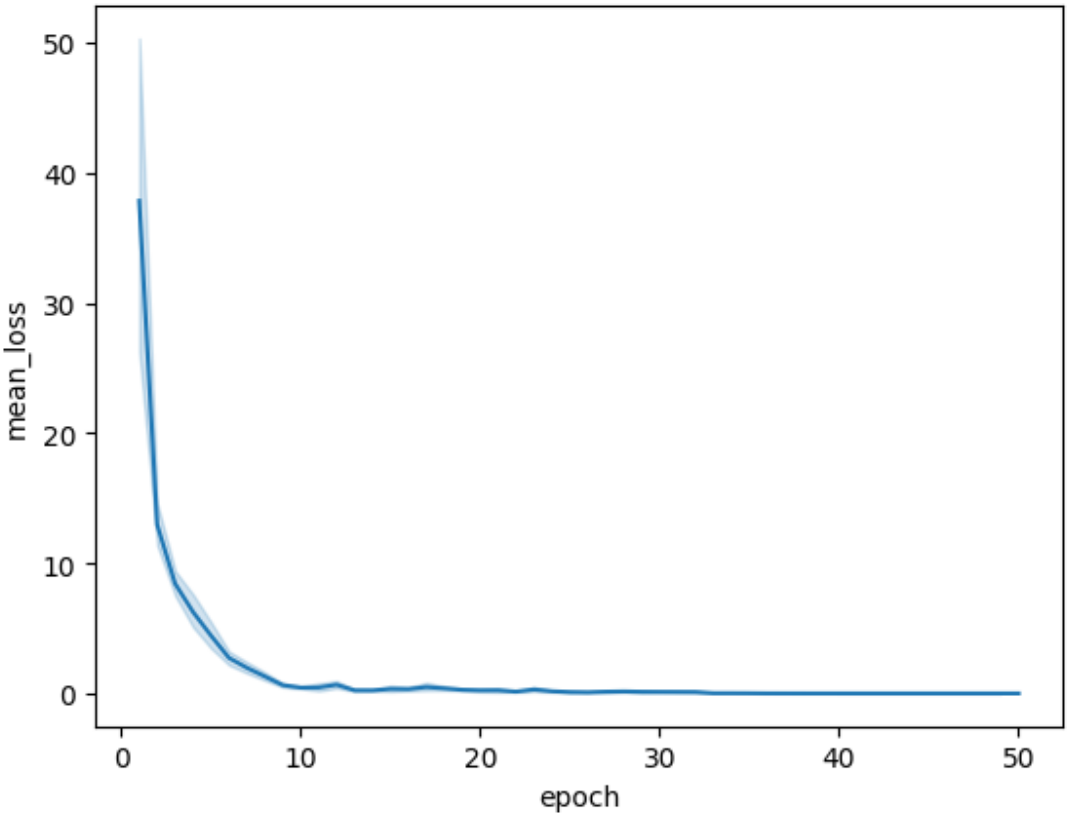

図 23：学習回数と平均損失の関係を表すグラフ

図 23 より、学習回数 10 回前後から平均損失が 0 に近づき、学習回数 25 回前後までは完全には安定せず、30 回を過ぎたあたりから細かな変動もなく平均損失がほとんど 0 に近い値を取り続けていることが読み取れる。つまりこのグラフから、精度の高いモデルは学習回数 30 回以上が必要であろうこと、学習回数が 15 回以上であれば精度が低くとも正しい出力形式が得られるであろうこと、学習回数が多ければ多いほど精度が高くなるであろうことが予測できる。改めて、本研究ではこの結果から学習回数 20, 30, 50 回のモデル(以下〇回のモデルと表記)を作成し、それぞれの評価とその精度比較を行った。

３．２　出力結果

3.2.1　出力に成功したモデル

学習した AI モデルを用いての出力では、出力のランダム性や AI の役割(ロール)の指示など出力に関する AI の設定が可能だが、本研究では特に設定の変更をせず各シーンのソース文を入力として出力を行ったところ正しい出力形式が得られた。設定可能な内容を含めた推論の実行コードとその出力結果を以下の図 24 に示す。

```
#モデル名を使ってGenerativeModelオブジェクトを生成
generative_model = genai.GenerativeModel(model_name=model_name)

#入力文の指定※必要であれば¥nで区切って出力の際のルールなどをAIに与える
prompt = """学習データに従ってください¥n¥nホームズは椅子に座った"""

#推論を実行(GenerativeModelオブジェクトでgenerate_contentを使用)
#※必要であればgeneration_configでランダム性のtemperatureなどを指定
response = generative_model.generate_content(
    prompt,
    generation_config={
        #出力のランダム性を指定
        "temperature": 1.0,
        #生成されるトークンの候補を確率の高い順に上位k個に絞り込む
        "top_k": 40,
        #生成されるトークンの候補を確率の高い順に累積確率がp以上になるまで絞り込む
        "top_p": 0.95,
        #生成される最大トークン数を指定
        "max_output_tokens": 1024
    }
)

# 結果を出力
print(response.text)
```

```
kdn:scene_x a kgc:situation ; kgc:hasPredicate kdp:sit ; kgc:subject kdn:Holmes ; kgc:what kdn:chair .
```

図 24：推論の実行コードと出力結果の一例(30 回のモデルを使用)

図 24 では 30 回のモデルを使った出力例を示しているが、20 回のモデルを用いた場合でも同様に正しい出力形式の結果を得られている。また先述した通り、実際の評価の際には出力のルールを追加する内容は加えておらず、generation_config 設定に変更は加えていない(図 24 内の generation_config 設定がデフォルトであり、コード内に表記しなくても内部でこのように設定されている)。尚、第 2 章で述べたように 50 回のモデルでは正しい形式の出力に失敗したため、20 回, 30 回のモデルに関する出力評価の後にその内容を記す。

上記のように正しい形式での出力が得られることを確認した後、学習データには用いなかった「まだらのひも」の残ったシーンのうち、50 シーンを評価データとして使用した。先述した Loadability, Precision, Recall, F1 の 4 つの観点で評価を行ったところ以下の表 1 のような結果が得られた。

表 1：20 回, 30 回のモデルの評価

epoch_count	Precision	Recall	F1	Loadability
20	0.684848	0.713	0.698801	1
30	0.815287	0.783	0.799046	1

3.1 のグラフから予側した通り、20 回のモデルより 30 回のモデルの方が精度が高い結果となったことがわかる。また、同義語の許容やデータセットの単純化を図ったことで、先行研究よりも十分に精度の高い評価結果を得られているが、それでも Precision や Recall が 0.9 を超えるような高い値にならなかったのはデータ数の少なさからだと考えられる。先行研究では 7 つの小説の約 2600 シーンを学習データとして用意したのに対し、本研究では 1 つの小説の 200 シーンを学習データとして用意したため、その数は約 1/13 であり種類自体も少ないといえる。学習量を約 10 倍にしたことで学習データと同じ「まだらのひも」に関するトリプル生成では精度の高い評価結果となったのであろう。仮に先行研究と同程度の学習量で本研究と同じ評価手法を用いれば、Precision, Recall ともに 0.9 を超えるような評価を得られる可能性があると考えられる。

3.2.2　出力に失敗したモデル

次に出力に失敗した 50 回のモデルに関して、実際の出力結果は以下の図 25 のようになった。

```
prompt = """ホームズは椅子に座った"""
response = generative_model.generate_content(prompt)

# 結果を出力
print(response.text)
```

```
a:sit ; ¥c:hasPredicate kdp:take ; ¥c:subject kdn:Holmes ; ¥c:what kdn:chair .
```

図 25：５０回のモデルを用いた出力の失敗例

図 25 におけるシーンは評価データに用いたものの中でもソース文がかなり単純(場所や時間などの目的語がなくトリプルが 3 つと少ない)なものの部類であるにもかかわらず、50 回のモデルでは正しい形式での出力に失敗した。この結果からわかるように他のシーンに対する出力でも正しい形式のものは得られず、評価基準を用いると Loadability が 0 という結果(それに伴い Precision, Recall, F1 も 0 と評価せざるを得ない)となった。先述した出力に関する AI の設定を変更してもやはり正しい形式の出力を得ることはできなかったが、AI に対するロールの設定を加えることで、図 25 と同様の単純なソース文に対しては一部が欠けているものの正しい形式の出力を得られたため以下の図 26 に示す。

```
prompt = """学習データに従って１文からトリプルを生成してください¥n¥nホームズは椅子に座った"""
response = generative_model.generate_content(prompt)

# 結果を出力
print(response.text)
```

```
kdn:hasPredicate kdp:sit ; kdn:subject kdn:Holmes ; kdn:where kdn:chair .
```

図 26：50 回のモデルを用いた、トリプルの不足を含む正しい形式での出力例

学習データに従うこと、1 文からトリプルを生成すること、という 2 点のロールを AI に与えたことで、最低限形式を守った出力が得られたと考えられる。

３．３　考察

まず 20 回, 30 回のモデルを比較した内容より、3.1 で示した平均損失のグラフ(図 23)から予測した性能の差が予測通りであったことがわかる。また先行研究のシーン数と学習回数の積(データ数約 2600×学習回数 3＝7800 とデータ数約 2600×学習回数 1＝2600)を含めて考えると、図 23 のグラフが示す通り学習回数とデータ数の積が 4000(本研究においてはデータ数 200×学習回数 20)の状態から正しい形式での出力が得られ、6000(本研究においてはデータ数 200×学習回数 30)更には 7800(先行研究)までは性能が向上していくと予測できる。

次に 50 回のモデルで正しい形式の出力が得られなかったことから、平均損失が 0 に近いことが必ずしも性能農高いモデルになるとは限らないことが読み取れる。そもそも平均損失はモデルの推論と実際のデータとの差を表す値であり、同じデータに対する差は学習回数に反比例することは容易に予測できる。学習回数の注意点は未知のデータに対する弱さとのバランスであり、多くのデータ数を少ない回数学習させるのと少ないデータ数を多く学習させるのでは、後者の方が未知のデータに対して弱いであろう。実際に 50 回のモデルで出力に失敗した原因は過学習であり、本研究では学習データの少なさを学習回数で補ったため、学習回数 50 回は多すぎたという結果になったのである。

ゆえに失敗例を考慮すると、本研究のようなデータを用いる場合は、学習回数とデータ数の積が 4000 前後の状態から正しい形式での出力が得られ、8000 前後まで性能が向上、それ以上では過学習により正しい形式での出力が得られなくなる、というように考えられる。尚、学習データが少なければ少ないほど小さい学習回数とデータ数の積の値過学習となるとも考えられる。

第4章　結論

4．1　　終わりに

本研究ではナレッジグラフ推論チャレンジの企画課題の一つである、小説の 1 シーンからトリプルを生成するというタスクに取り組み、第 1 章 1.4 で示した先行研究をもとに学習データの作成・ファインチューニング・出力評価を行い、精度高い AI モデルの作成を目指した。

結果・考察より、ファインチューニングをするうえで学習回数等の設定は試行を重ねて正確に調整する必要があることがわかった。学習不足や過学習により、入力文に対して適切な出力が判断できない、入力文が学習したデータにはないため適切に認識できないという状況が AI に起こるのである。またそれらを避けるためにも学習データの幅広さは重要であり、データの種類の豊富さや十分な量のデータ数、適切なファインチューニングの設定があり初めて精度の高い AI モデルを作成できることが確認できた。

ナレッジグラフ推論チャレンジの全工程をふまえても、トリプル生成の精度向上は重要である。正確なトリプルを生成することで正確なナレッジグラフを構築でき、それが正確な推論につながると考えられる。

4．2　　今後の展望

本研究では比較対象として用いなかった、学習データの数や種類、入力文の書き方を変えての精度比較などを行うことで、より細分化されたモデル性能の差を理解することができる。それにより、より汎用性の高いモデルを作成することができ、ナレッジグラフ推論チャレンジでは指定されていないシャーロック・ホームズシリーズ以外の小説にも推論が可能になると考えられる。また、入力を文だけでなく画像や音声にまで拡大することで漫画やアニメにまで推論を可能とし、幅広い分野で活用できるシステムにすることができるとも考えられる。ナレッジグラフ推論チャレンジをはじめとする、生成 AI の応用機会には注目していきたい。

参考文献

[1]NEC, 2024, ナレッジグラフ推論チャレンジホームページ, (2025 年 2 月 22 日取得, https://www.nec-solutioninnovators.co.jp/sp/contents/column/20240229_llm.html)
[2]人工知能学会・セマンティックウェブとオントロジー研究会・企画委員, 2024, ナレッジグラフ推論チャレンジホームページ, (2025 年 2 月 22 日取得, https://challenge.knowledge-graph.jp/2024/index.html)
[3]人工知能学会・セマンティックウェブとオントロジー研究会・企画委員, 2024, SPARQL エンドポイント, (2025 年 2 月 22 日取得, http://knowledge-graph.jp/sparql2020v2.html)
[4]川村隆浩, 江上周作, 田村光太郎, 外園康智, 鵜飼孝典, 小柳佑介, 西野文人, 岡嶋成司, 村上勝 彦, 高松邦彦, 杉浦あおい, 白松俊, 張翔宇, 古崎晃司, 2019, 第 1 回ナレッジグラフ推論チャレ ンジ 2018 開催報告—説明性のある人工知能システムを目指して—, (2025 年 2 月 23 日取得)
[5]人工知能学会・セマンティックウェブとオントロジー研究会・企画委員, 2024, ナレッジグラフ可視化ツール, (2025 年 2 月 23 日取得, http://knowledge-graph.jp/visualization/)
[6]富士通株式会社, 2023, 「OpenAI API を使った自然言語からの KG 構築」, (2025 年 2 月 23 日取得, https://github.com/KnowledgeGraphJapan/KGRC-ws-2023/blob/main/swo60.pdf)