【調査研究】

LLM の評価手法の提案と学習支援効果

安藤優希・冨山哲男・山下一郎 *

Proposal for LLM Evaluation Methods and Learning Support Effectiveness

Yuki Ando, Tetsuo Tomiyama and Ichiro Yamashita*

Abstract: This research study aims to evaluate both the natural language understanding and code generation capabilities of large-scale language processing quantization models, to determine their feasibility as learning aids on student PCs. The method involves using zero-shot prompting, i.e., inputting prompts without prior information, on problems posted on the competitive programming site AtCoder. The generated code will be submitted, and its accuracy and quality will be assessed. The outcome suggests that for complex problems, models tuned to instructional tasks, known as instruction models, are likely to be more effective in aiding learning.

Keywords: Large Language Models, Quantization, Artificial Intelligence

1　概要

本調査研究では、大規模言語処理量子化モデルの自然言語理解度とコード生成能力の両方を検証し、学生用 PC 上で学習の補助として使用可能なのかを調査するための研究である。手法は、競技プログラミングサイト AtCoder に掲載されている問題を zero-shot prompting、つまり、事前情報なしで prompt 入力し、生成されたコードを提出し正答率や生成されたコードから評価を定める。成果では , 高度な問題であるならば命令タスクにチューニングされた instruction モデルの学習支援効果が高いと考えられる。

2　序論

今日自然言語処理の分野での進歩により、ChatGPT などの AI モデルが高度な理解能力と文章生成能力を発揮している。例えば、個人用の PC でも動作するように、メモリ使用量を減らし、推論速度を向上させても精度があまり低下しない 4bit 量子化が提案されている [8, 10, 11]。このような技術の進歩は、小中等教育や研究、ホワイトカラーの業務に新たな可能性をもたらしており、その有効活用について様々な議論が行われている [1, 2]。

* 株式会社オレンジテクラボ

また、同時に複雑なセクションを除き文書作成の支援では、人間による校正が必要ではあるが使用することができるという研究も出ている [12]。

本調査研究では、Python や java、Shell などのコードを LLM がターミナル上で ChatGPT のようなインターフェイスを介して、ローカルで実行することができる open interpreter を用い、自然言語の理解度と生成能力の両方を評価し、学習の補助に適しているモデルを調査するための研究である。また、モデルのチューニングの違いによって得意な言語や傾向が異なり、使用するライブラリによって生成されるコードが大きく変化し生成能力を正しく比較できない為、数理問題の中でも外部ライブラリを使用せずに解ける問題を取り扱い、モデルの学習データの種類によって生成されるコードの生成能力を比較し、学習の補助に最適なモデルを調査するための研究である。

3 方法論

本調査研究では、競技プログラミングサイト AtCoder の DP(動的計画法) を練習できる「Educational DP Contest / DP まとめコンテスト」から選定した問題を open interpreter を用いて、zero-shot prompting、つまり事前情報なしで prompt 入力を行いコード生成結果よりモデルの評価を行う [3]。また，推論タスクにおいてステップごとに考えるといった命令を含めることで精度が向上するが今回は使用しないことにする [9]。また、token 数が大きくなると LLM 側の指示の遵守率が低下するため、1 文字 1~3 token の日本語ではなく、基本的に 1 単語 1 token として扱う英語に翻訳しこの token 数増大による精度の低下を防ぎ、太郎君やカエルといった文章題特有の一般名詞などにモデルの理解が引っ張られないようにするため、これら一般名詞を無くした状態を prompt として扱う [4]。検証として出力されたコードを実際に AtCoder に提出することで、正答率をモデル評価に考慮する。

3.1 問題選定と Prompt 例

本調査研究では Local モデルにおける生成能力と自然言語理解の評価に重きを置いているので、評価を容易にするため DP 問題でも比較的容易な決定や計算が一連の順序付けられたステップに依存しており、DP テーブルの配列構造が単純な AtCoder の A 〜 E 問題までに絞り、絶対値を用いた最小値の最適化問題である A 問題、選択肢に基づく最大値の最適化問題である C 問題、DP 問題として有名なナップサック問題である D 問題を選定した [5]。

B、E 問題では A、D 問題と同一の内容から制約のみが変更され、より難易度が高くなっているため、本調査研究での評価が不明な状態では不適切であると考えるため除外した。

表 1 に本調査研究で使用した prompt 文を挙げる。

表 1：使用 prompt 文一覧

問題番号	prompt 文
A	Create Python code based on the following problem statement. Problem Statement: There are N platforms, each numbered from 1 to N, with heights denoted as h_1, h_2, ..., h_N. Initially, you are on platform 1, and you want to reach platform N by repeating the following action multiple times: When you are on platform i, you can jump to either platform i+1 or platform i+2. When you make this jump, you incur a cost equal to the absolute difference in heights between platform i and platform j, denoted as |h_i - h_j|.
C	Create Python code based on the following problem statement. Problem Statement: There are N days. For each i (1≤i≤N), you can choose one of the following options: A: Gain happiness a_i. B: Gain happiness b_i. C: Gain happiness c_i. You cannot choose the same option for two or more consecutive days. Please calculate the maximum total happiness.
D	Create Python code based on the following problem statement. Problem Statement: There are N items. Each item is numbered 1, 2, …, N. For each i (1 ≤ i ≤ N), item i has a weight w_i and a value v_i. You have decided to select some of these N items to take back with you. The capacity is W, and the total weight of the items you take back must not exceed W. Find the maximum total value of the items you can take back.

3.2　使用モデルについて

今回使用する local モデルは Hugging Face という機械学習モデルの開発と共有、公開をするためのプラットフォームから選定し使用する [6]。使用するモデルのフォーマットは 4bit 量子化されており、品質の低下が少ないとされている Q4_K_M の .GGUF フォーマットを用いる。また、モデルの種類は Llama2 というテキスト生成や理解 (対話ユースケース) に最適化されたモデルを、コード生成に特化させた学習データを用いチューニングした CodeLlama のベースモデル、命令タスク (自然言語の理解) に特化させたチューニングをした instruction モデル、Python に特化させたチューニングをした Python モデルの三種類の 7B、13B、34B といったパラメータ数の違うモデルを用いることとする。CodeLlama とは Meta 社が開発したコード生成専用の LLM であり、5000 億 token のプログラムに特化させた学習データを使用しているモデルのことである。また、このモデルにできることはプログラムの補完、prompt からプログラミング、プログラム関連の質問に答えるという 3 種類のことが可能である。

表 2：使用モデル例一覧

ベースモデル	instruction モデル	Python モデル
CodeLlama 7B	CodeLlama 7B	CodeLlama 7B
CodeLlama 13B	CodeLlama 13B	CodeLlama 13B
CodeLlama 34B	CodeLlama 34B	CodeLlama 34B

3.3 モデル評価手法

モデルの検証方法には、AtCoder に掲載されている「Educational DP Contest / DP まとめコンテスト」から足場という座標から移動する際のコストを最小限にする最適化問題の A 問題、選択肢に基づく最大値の最適化問題である C 問題、DP 問題として有名なナップサック問題である D 問題を用い、実際にコンテストに提出することで出た正答率とコード内容から独自の評価基準で評価する。また、提出に際して標準入力はユーザの手で加工するものとし、関数や導出処理部分に関してのみ評価の焦点をあてる。

独自の評価基準に関しては学生用 PC で動かし学習の補助ということを考慮し、「モデルの規模 (Model Size)」、「生成スピード (Generation Speed)」、「意味的正確性 (Semantic Accuracy)」、「文法的正確性 (Grammatical Accuracy)」、「問題対応力 (Problem Solving Ability)」の 5 つから見るものとする。

最大スコアを 5 とし、モデル規模はモデルごとの相対評価、生成スピードはモデルごとの絶対評価、意味的正確性や文法的正確性は減点方式の評価、問題対応力は AtCoder の正答率をもとに計算し評価する。以下に AtCoder 評価基準と独自の評価基準を用いた評価例を挙げる [7]。

3.3.1 AtCoder 評価基準

AtCoder では問題に対する解法のコードを提出することでサンプルケースが与えられ、そのサンプルケースにおいて起きた現象を以下の判定で評価する [7]。

表 3：AtCoder 判定基準一覧

AC	運営が用意したテストをパスし，正しいプログラムと判定
WA	プログラムは動いたが出力が違う
RE	コンパイル時に検出できなかったエラーが発生
CE	コンパイルエラー
IE	ジャッジシステムのエラー
TLE	指定された実行時間を超過
MLE	指定されたメモリの超過
OLE	指定されたサイズを超過

3.3.2 モデルの規模 (Model Size)

検証モデルの Model Size 種類によって値の取り方が変化する。また、この評価は 1 〜 5 の範囲で評価される。更に、Model Size が一番小さいものを 5 としそこから相対評価をとる。本調査研究では 7B が 4.08 GB、13B が 7.87 GB、34B が 20.22 GB の三種類であり、7B を 5 とした場合、13B を 3、34B を 1 とした。

3.3.3 生成スピード (Generation Speed)

モデルの平均的な生成時間に応じて評価する。この評価もモデルの規模同様 1 〜 5 の範囲で評価される。また、ループに陥った場合はループ発生時点までを生成スピードとして

扱うものとする。

評価基準は表 4 の通りである。

表 4：生成スピード評価基準一覧

Time	Score
~10 秒	5
10 秒~15 秒	4
15 秒~20 秒	3
20 秒~30 秒	2
30 秒以上	1

3.3.4　意味的正確性 (Semantic Accuracy)

減点方式に関しては基礎スコアを生成コードあたりのエラー数とし、生成されたコード数を考慮し評価を変化させたいため 0 ～ 1 の範囲で基礎スコアに重みを付与したスコアを加算する。以下に数式例を挙げる。

$$\frac{E}{C}+\frac{E}{C}\left(1-\frac{logC}{logN}\right)$$

ⅰ．　$1-\frac{logC}{logN}$: コードの生成量によって変化する重み

ⅱ．　C: 生成されたコード数（$1 \leq C \leq N$）

ⅲ．　E: 問題文を理解できていないコード (生成無し , 不適や全探索など) 数とする

ⅳ．　N: 問題総数（$1 \leq N$）

ⅴ．　コード出力なしや不適しかないもの、エラー数が極端に多く負の値になってしまう評価値は意味的正確性、文法的正確性を 0 評価とする。

3.3.5　文法的正確性 (Grammatical Accuracy)

意味的正確性と同様の数式を用いる。条件の変更点は以下である。

ⅲ．　E: エラーの原因数

3.3.6　問題対応力 (Problem Solving Ability)

ⅰ．　m: WA, TLE, MLE, OLE の数

ⅱ．　N: 問題総数とする

ⅲ．　tは任意の数値であり、全てWAだった場合の最低スコアを表している。今回はエラー毎に評価値を変え、評価値を明瞭にするため差を均等にし、WA 数の影響をなるべく受けるよう調整し 0.3 とする。

表 5：問題対応力評価基準

AC	1
WA/TLE/MLE/OLE	$t + (1 - t) \times (1 - \frac{m}{N})$
RE	0. 2
CE	0. 1
生成不適，無し	0

3.4 AtCoder サンプル問題傾向

AtCoder にはユーザの問題への理解度や解決方法を確認するためサンプルケースが与えられ、それぞれのケースごとの出力結果によって判定が変化する。また、問題文にはプレサンプルデータが掲載されており、ユーザはこのサンプルデータを基にコードを記述していく。

3.4.1 AtCoder A 問題

準入力では足場の量 N が最初に与えられ、次に足場の高さ h が与えられる。

制約は $2 \leq N \leq 10^5$ 、 $1 \leq h_i \leq 10^4$ 。

表 6：AtCoder A 問題サンプルケース傾向一覧

ケース名	サンプル問題傾向
0_00	プレサンプルデータと同一
0_01	プレサンプルデータと同一
0_02	プレサンプルデータと同一
1_00	N が小さい. h がすべて同じ値
1_01	N が小さい. h がとれる値の最少と最大
1_02	N がとれる最大の値−1. h がとれる値の最少と最大を交互
1_03	N がとれる最大の値. h がとれる値の最少と最大をパターン化
1_04	N が大きい. h はとれる値内でランダム
1_05	N が大きい. h はとれる値内でランダム
1_06	N が大きい. h はとれる値内でランダム
1_07	N が大きい. h はとれる値内でランダム

3.4.2 AtCoder C 問題

標準入力では休みの総日数 N が与えられ、N の値に応じて各日の幸福度 3 種類 (a, b, c) が与えられる。

制約は $1 \leq N \leq 10^5$ 、$1 \leq a_i, b_i, c_i \leq 10^4$ 。

表 7：AtCoder C 問題サンプルケース傾向一覧

ケース名	サンプル問題傾向
0_00	プレサンプルデータと同一
0_01	プレサンプルデータと同一
0_02	プレサンプルデータと同一
1_00	N がとれる最少の値. a，b，c がすべてとれる最大の値
1_01	N とすべての a，b，c が，とれる最大の値
1_02	N が大きい. a，b，c はとれる値内でランダム
1_03	N が大きい. a，b，c はとれる値内でランダム
1_04	N が大きい. a，b，c はとれる値内でランダム
1_05	N が大きい. a，b，c はとれる値内でランダム
1_06	N が大きい. a，b，c はとれる値内でランダム
1_07	N が大きい. a，b，c はとれる値内でランダム
1_08	N が大きい. a，b，c はとれる値内でランダム
1_09	N が大きい. a，b，c はとれる値内でランダム

3.4.3　AtCoder D 問題

標準入力ではものの数 N と重さの総和 W が与えられ、N の値に応じて順次重さ w と価値 v が与えられる。

制約は $1 \leq N \leq 100$ 、$1 \leq W \leq 10^5$ ，$1 \leq w_i \leq W$ ，$1 \leq v_i \leq 10^9$ 。

表 8：AtCoder D 問題サンプルケース傾向一覧

ケース名	サンプル問題傾向
0_00	プレサンプルデータと同一
0_01	プレサンプルデータと同一
0_02	プレサンプルデータと同一
1_00	N がとれる最少と W がとれる最大の値. w，v がとれる最大の値
1_01	N，W，w，v がとれる最大の値
1_02	N がとれる最大の値. W，w，v はとれる値内の大きい値でランダム
1_03	N がとれる最大の値. W，v はとれる値内の大きい値でランダム. w はとれる値内でランダム
1_04	N がとれる最大の値. W，v はとれる値内の大きい値でランダム. w はとれる値内でランダム
1_05	N がとれる最大の値-1. W，v はとれる値内の大きい値でランダム. w はとれる値内でランダム
1_06	N がとれる最大の値. W，v はとれる値内の大きい値でランダム. w はとれる値内でランダム
1_07	N がとれる最大の値-1. W，v はとれる値内の大きい値でランダム. w はとれる値内の小さい値でランダム
1_08	N がとれる最大の値-1. W，v はとれる値内の大きい値でランダム. w はとれる値内の小さい値でランダム
1_09	N がとれる最大の値. W，v はとれる値内の大きい値でランダム. w はとれる値内の小さい値でランダム

4 結果

4.1 出力されたコード例

4.1.1 AtCoder A 問題

全体的に絶対値の扱い方をよく理解できていないのか、図 2 のように if 分岐の下で行われている絶対値の加算が同一の計算をおこなっているなど、条件分岐が意味をなしていないものや図 1 のように計算に必要な情報を利用していないものが見受けられた。

```python
def jumping_on_platforms(heights):
    # Initialize variables
    n = len(heights)
    dp = [[0] * (n + 1) for _ in range(2)]

    # Fill in the table
    for i in range(1, n + 1):
        for j in range(i + 1, n + 1):
            dp[i % 2][j] = min(dp[(i - 1) % 2][j], dp[(i - 2) % 2][j - 1])

    # Return the optimal cost
    return dp[n % 2][n]
```

図 1：CodeLlama 34B instruction

```python
import sys

def jumping_cost(heights):
    cost = 0
    for i in range(len(heights) - 1):
        if heights[i] > heights[i + 1]:
            cost += abs(heights[i] - heights[i + 1])
        elif heights[i] < heights[i + 1]:
            cost += abs(heights[i + 1] - heights[i])
    return cost
```

図 2：CodeLlama 13B instruction

4.1.2 AtCoder C 問題

ほとんどのモデルでコードの出力が不十分であった。また、図 3 のように DP テーブルを作成しているにも関わらず、その後の条件分岐で制約を無視したコードや、図 4 のように最初から制約を無視するなど、制約の無視が多く見受けられた。

```python
def max_happiness(a, b, c):
    n = len(a)
    dp = [[0] * (n + 1)] * (n + 1)
    prev = [-1] * (n + 1)
    dp[0][0] = a[0]
    prev[0] = 0

    for i in range(1, n):
        option_a = dp[i - 1][i - 1] + a[i]
        option_b = dp[i - 2][i - 2] + b[i]
        option_c = dp[i - 3][i - 3] + c[i]

        if option_a >= option_b and option_a >= option_c:
            dp[i][i] = option_a
            prev[i] = 0
        elif option_b >= option_a and option_b >= option_c:
            dp[i][i] = option_b
            prev[i] = 1
        else:
            dp[i][i] = option_c
            prev[i] = 2

    return dp[n - 1][n - 1]
```

図 3：CodeLlama 34B instruction

```python
# Define the happiness values for each option
happiness_A = [a_1, a_2, a_3, a_4, a_5]
happiness_B = [b_1, b_2, b_3, b_4, b_5]
happiness_C = [c_1, c_2, c_3, c_4, c_5]

# Define the maximum total happiness
max_total_happiness = 0

# Iterate through each day and calculate the maximum total happiness
for i in range(N):
    # Calculate the maximum total happiness for each day
    max_total_happiness = max(max_total_happiness, happiness_A[i] + happiness_B[i] + happiness_C[i])

# Print the maximum total happiness
print("The maximum total happiness is", max_total_happiness)
```

図 4：CodeLlama 7B instruction

4.1.3　AtCoder D 問題

DP 有名問題ということもあってか、DP 法を使用したコードが多く存在した (図 5)。また、DP 法を使っていないコードは図 6 のように、制限まで取得し続けるものや重さと価値の比率で取得するものであった。

```python
def knapsack(weights, values, capacity):
    # Initialize variables
    n = len(weights)
    dp = [[0] * (capacity + 1) for _ in range(n + 1)]

    # Fill in the table
    for i in range(1, n + 1):
        for j in range(1, capacity + 1):
            if weights[i - 1] > j:
                dp[i][j] = dp[i - 1][j]
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weights[i - 1]] + values[i - 1])

    # Return the optimal value
    return dp[n][capacity]
```

図 5：CodeLlama 34B instruction(DP)

```python
import sys

def get_max_value(weights, values, capacity):
    # Initialize variables
    max_value = 0
    current_weight = 0
    current_value = 0

    # Sort the items by value/weight ratio in descending order
    sorted_items = sorted(zip(values, weights), key=lambda x: x[1]/x[0], reverse=True)

    # Iterate through the items and add them to the knapsack until the capacity is exc
    for value, weight in sorted_items:
        if current_weight + weight <= capacity:
            max_value += value
            current_weight += weight

    return max_value
```

図 6：CodeLlama 13B instruction(比率)

4.2 それぞれのモデル差異

4.2.1 ベースモデル

コード出力は 2 つのモデルしか出力せず、4.2.1.1 CodeLlama 13B(A 問題) のように問題の意図を理解していないものと 4.2.1.2 CodeLlama 13B(D 問題) のように理解しているものと極端な生成が見受けられ不安定であった。

```python
def f(n):
    return n * (n + 1) / 2
```

図 7：CodeLlama 13B(A 問題)

```python
def max_value(W, wt, val):
    n = len(wt)
    K = [[0 for x in range(W + 1)] for x in range(n + 1)]
    for i in range(n + 1):
        for w in range(W + 1):
            if i == 0 or w == 0:
                K[i][w] = 0
            elif wt[i - 1] <= w:
                K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w])
            else:
                K[i][w] = K[i - 1][w]
    return K[n][W]
```

図 8：CodeLlama 13B(D 問題)

4.2.2　instruction モデル

他モデルよりも DP 法らしき手法をとっているものが多かったが、図 3 のように DP テーブルの初期化不足や不必要な変数の使用、図 9 のように if 分岐後の処理に範囲外のインデックスを選択する可能性を残していたりなど初歩的なミスが多く見受けられた。

```python
def min_cost(heights):
    if len(heights) == 0:
        return 0
    elif len(heights) == 1:
        return 0
    else:
        cost1 = abs(heights[0] - heights[1]) + min_cost(heights[2:])
        cost2 = abs(heights[0] - heights[2]) + min_cost(heights[1:])
        return min(cost1, cost2)

print(min_cost([h_1, h_2, ..., h_N]))
```

図 9：CodeLlama 7B instruction(A 問題)

4.2.3　Python モデル

Python モデルでは具体的なコードは出力せず、4.2.3.1 CodeLlama 7B Python(D 問題) のように関数の型だけ書いて終わるものがほとんどであった。

```python
def max_value(W, w, v):
    # TODO: write function here
    return None
```

図 10：CodeLlama 7B Python(D 問題)

4.3 AtCoder による評価結果

表 9：AtCoder による評価結果一覧

↓問題/モデル名→	CodeLlama 7B	CodeLlama 13B	CodeLlama 34B	CodeLlama 7B inst	CodeLlama 13B inst	CodeLlama 34B inst	CodeLlama 7B Python	CodeLlama 13B Python	CodeLlama 34B Python
A	RE			RE	7	9			
C				13		10			
D		0		RE	8	0			

AC
WA
RE
TLE
出力無し

※表内の数値は AC 以外の数

以上の結果から C 問題がもっとも正答率、生成率が低いことがわかり、A 問題は多様な判定結果が見受けられる。また、D 問題は生成されたコードの正答率は他問題よりも高い傾向にあった。

4.4 独自による評価結果

表 10：独自による評価結果一覧

	CodeLlama 7B	CodeLlama 13B	CodeLlama 34B	CodeLlama 7B inst	CodeLlama 13B inst	CodeLlama 34B inst	CodeLlama 7B Python	CodeLlama 13B Python	CodeLlama 34B Python
Model Size	5.0	3.0	1.0	5.0	3.0	1.0	5.0	3.0	1.0
Generation Speed	5.0	4.0	4.0	5.0	3.0	3.0	5.0	5.0	5.0
Semantic Accuracy	1.0	3.631	0.0	3.666	3.631	4.667	0.0	0.0	0.0
Grammatical Accuracy	3.0	5.0	0.0	4.333	4.315	3.667	0.0	0.0	0.0
Problem Solving Ability	0.2	1.0	0.0	0.7	1.124	1.889	0.0	0.0	0.0

以上の結果から汎用的なベースモデルには意味的正確性に大きなばらつきがみられた。また、文法的正確性に関してはどちらもばらつきがみられ、そこまでコード生成能力に差が見られないと考えられる。問題対応力では平均して instruction モデルが高い水準を誇っていた。

図 11 にレーダーチャートを表示する。

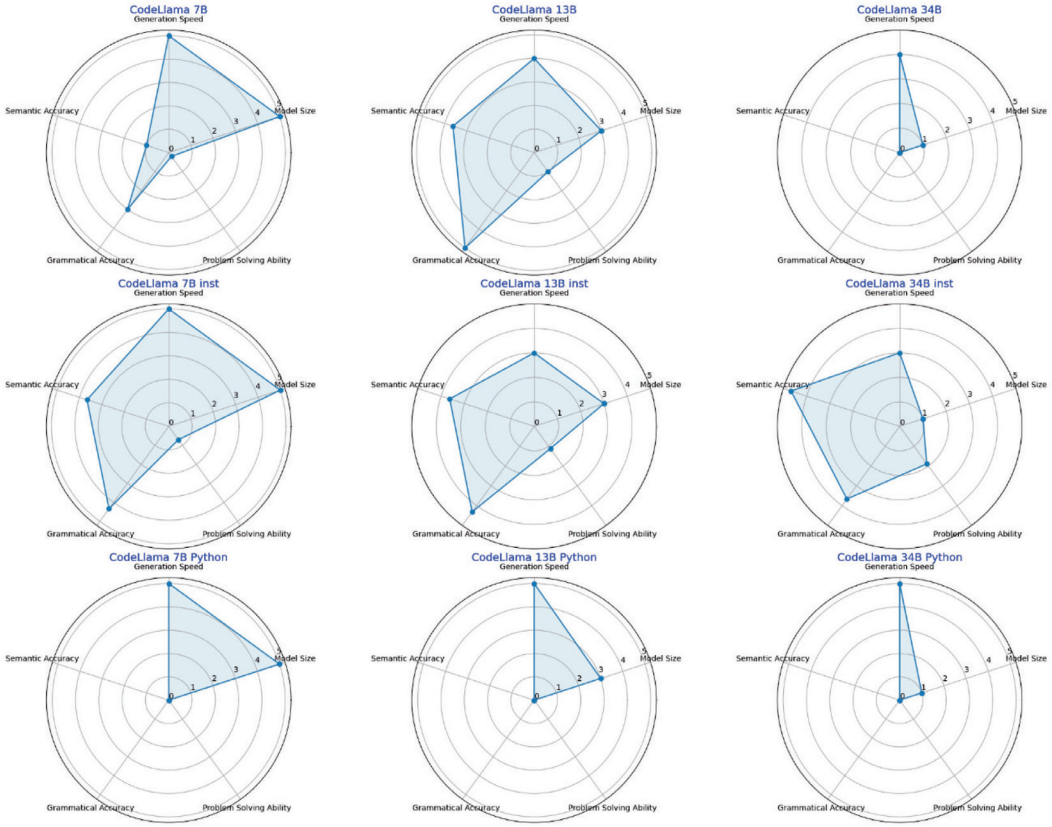

図 11：独自評価による結果をレーダーチャートにしたもの

5　考察

AtCoder による評価結果では、instruction モデルのような自然言語理解に特化させたモデルにおいて、ナップサック問題といった複雑で有名な問題は学習データに含まれている可能性があり、平均的な生成精度が高いのだと考えられる。

また、instruction モデルでは平均的な生成量は高いが正答数を見てみるとベースモデルに比べ、生成精度がモデルサイズに依存していることがわかる。このことから instruction モデルでは他モデルよりもパラメータ数が生成精度に大きな影響を与えていると考えられる。

独自の評価結果では、AtCoder と違った評価を得ることができた。例えば同規模の CodeLlama 13B と CodeLlama 13B instruction では、表 9 を見比べるとベースモデルの CodeLlama 13B の方が完全正答を出しており、instruction モデルと比べ生成量も同じであり間違えた数も少ないが、ベースモデルは完全正答と全不正解と二極化した結果となった。しかし、instruction モデルでは正答率こそ低いものの問題別に同じような正答率を出していることからモデルの安定性や汎化能力という面では instruction モデルが優れていると考える。そのため、表 5 はこの基準でも適切な評価を得られたと考える。

また、意味的正確性や文法的正確性はユーザの問題やコードへの理解度やユーザの判断

基準に依存してしまうため、判定基準を検証ごとに定める必要があると考える。更に、エラーの重大度などを考慮していないためこの評価基準は見直しが必要であると考えられる。

最後に、今回の研究では Python モデルの出力はすべての問題を通して図 10 のように出力不適、または無しであった。しかし、機械学習のテンプレート作成などにおいては import してくるモジュールなどに違いがあったため、数理アルゴリズムというよりも機械学習コードの作成や添削に特化させているモデルであると考えられる。

6 結論

モデル差が見受けられたものの、一部のモデルでは学習の補助としては使えると考える。また、自然言語理解に特化させた instruction モデルではほとんどのモデルが何かしらのコードを出力しており、平均的なモデル精度は比較的高かった。このことから、数理学習の補助に関しては instruction モデル且つパラメータ数の多いモデルのほうが良いと考える。

また、モデルの精度が非常に高いものは見受けられなかったが、ある程度のテンプレートは作成されていたので、被学習者が正しい知識を保持していれば生成コードを参考に添削可能であった。

7 謝辞

本研究を進めるにあたり、貴重な指導と実験に関するアドバイスを下さった西田健次 CTO 様、山田シニア UX デザイナー様、宮﨑 淳 東京国際工科専門職大学 客員教授、オレンジテクラボ CEO 様に深く感謝いたします。

参考文献

[1] 生成 AI の利用について：文部科学省 (mext.go.jp), 2023 年 7 月 4 日 . accessed December 10, 2023.
https://www.mext.go.jp/a_menu/other/mext_02412.html

[2] 経済産業省 . 生成 AI 時代の DX 推進に必要な人材・スキルの考え方 . デジタル時代の人材政策に関する検討会 , 2023 年 8 月 7 日 . accessed December 10, 2023.
https://www.meti.go.jp/press/2023/08/20230807001/20230807001.html

[3] Educational DP Contest / DP まとめコンテスト , accessed October 23, 2023.
https://atcoder.jp/contests/dp

[4] Anthropic Claude2 で Prompt 実行の精度を高めるテクニック , accessed October 30, 2023.
https://qiita.com/nkitaarashi/items/85ac0eb7c451f30972d4

[5] Educational DP Contest の F 〜 J 問題の解説と類題集 , accessed December 26, 2023.
https://qiita.com/drken/items/03c7db44ccd27820ea0d

[6] hugging face の紹介 , accessed December 28, 2023.
https://zenn.dev/yoshikawat64m/articles/6c7862ad376368

[7] 競技プログラミング用語集 , accessed December 28, 2023.
https://atcoder.jp/contests/abc074/glossary?lang=ja

[8] LIU, Zechun, et al. LLM-QAT: Data-Free Quantization Aware Training for Large Language Models. In International Conference on Learning Representations (ICLR), September 19, 2023. accessed January 18, 2024.
[9] KOJIMA, Takeshi, et al. Large language models are zero-shot reasoners. Advances in neural information processing systems, 2022, 35: 22199-22213. accessed January 18, 2024.
[10] LIU, Jing, et al. QLLM: Accurate and Efficient Low-Bitwidth Quantization for Large Language Models. In International Conference on Learning Representations (ICLR), January 16, 2024. accessed January 18, 2024.
[11] FRANTAR, Elias, et al. OPTQ: Accurate Quantization for Generative Pre-trained Transformers. In International Conference on Learning Representations (ICLR) poster. February 02, 2023. accessed January 19, 2024.
[12] MICHELET, Gaëtan; BREITINGER, Frank. ChatGPT, Llama, can you write my report? An experiment on assisted digital forensics reports written using (Local) Large Language Models. In Digital Forecsics Research Conference (DFRWS EU). December 29, 2023. accessed January 19, 2024.

安藤優希　東京国際工科専門職大学　工科学部　情報工学科　3 年
冨山哲男　東京国際工科専門職大学　工科学部　情報工学科　教授
山下一郎　株式会社オレンジテクラボ