

Coq と大規模言語モデルによる相互学習

鈴木彩加 *

概要

本論文では，定理証明支援系 Coq と大規模言語モデルを組み合わせた定理証明の自動化手法を提案する．まず，Coq のスクリプトとそれに対応するゴール，次に適用すべき tactic のデータセットを作成した．このデータセットで大規模言語モデルに追加学習を行い，学習したモデルを用いて Coq の tactic を生成し検証した．本手法により，形式言語を使用した定理証明を目的とした大規模言語モデルへの追加学習が効果的であることを示した．

目次

1	はじめに	2
2	関連研究の概要	4
2.1	定理証明支援系	4
2.2	大規模言語モデル	7
3	問題設定	12
3.1	相互学習	13
4	手法	14
4.1	実験環境	15
4.2	パラメータ	15
5	実験 I	17
5.1	データセット	17
5.2	実験設定	19
5.3	実験結果	19
5.4	考察	22
6	実験 II	22

* 明治大学大学院理工学研究科数学専攻, Email: s1suzukisayaka1s@gmail.com

6.1	データセット	22
6.2	実験設定	25
6.3	検証方法	27
6.4	実験結果	27
6.5	考察	30
7	実験 III	31
7.1	データセット	31
7.2	実験設定	32
7.3	検証方法	32
7.4	実験結果	33
7.5	考察	33
8	まとめ	35
8.1	考察	36
8.2	課題	37
8.3	将来研究	37
A	付録	41
A.1	追加学習, 推論に関するソースコード	41
A.2	第 5 章の実験 I に関するソースコード	47
A.3	第 6.1 節のデータセット作成に関するソースコード	49
A.4	第 6 章の実験 II に関するソースコード	54
A.5	モジュール	55
A.6	その他	55

1 はじめに

数学において証明を行うことは中心的な作業である。長くて複雑な証明を行うこと、またそれを確認することは、時間がかかる作業であり、ミスも起こりやすい。そこで数学の証明およびその確認を機械により自動化する定理証明支援系と呼ばれるものが開発されてきた。よく知られている定理証明支援系としては、Coq や Lean, Agda などがある。定理証明支援系は、対話型の特殊なプログラミング言語で数学の形式的な定理証明の検証を支援する。

ところで最近では ChatGPT などの大規模言語モデルの発展が目覚ましい。大規模言語モデルに数学の問題を解かせる研究も盛んに行われている。一方、大規模言語モデルに

は hallucinations (幻覚) の問題があり、堂々と間違っただけを答えることがある。

本研究では、言語モデルと定理証明支援系を組み合わせることで、機械検証された数学の定理の証明を与えることを目指す。定理証明支援系は正しいかどうかを確認することが得意だが、証明の大きな流れを掴むことは難しい。一方、言語モデルでは大規模データセットから学習を行うことができるが、細かい計算は苦手なようである。そこで、言語モデルに定理証明支援系の証明データを学習させ、次の命令を予測させる一方、定理証明支援系にその妥当性を検証させる。このように相互に情報を与え合うことで、機械検証され正しいことが保証された証明を機械に出力させることを目指す。

証明を作成する中で、証明の各段階の状態にどの段階に進むにはどうすると良いのか (どの tactic を指定すれば良いのか) が決定できるかどうか重要である。本研究では、証明のある段階の状態に対して、最適な tactic を言語モデルが出力できるか検証した。

関連する先行研究として以下がある。既存の大規模言語モデルに関する研究では、自然言語によるテキスト文やプログラムなど単一の要素のみを対象とした研究が盛んである。また、数学の分野では、大規模言語モデルに自然言語で数学の問題を解かせる際に自然言語の数学のデータセットで学習した研究が盛んである (例えば, [8, 9])。

定理証明と深層学習の組み合わせについての考察がなされている [11]。Coq の tactic 予測に関する既存研究では、ルールベースのものや初期のニューラルネットワーク等を使用している [4]。今回の提案手法は大規模言語モデルを使用している。定理証明支援系 Lean では Lean-dojō [15] をはじめとして大規模言語モデルを使った研究が進んでいる。

本研究では実験を 3 つ行った。本論文の構成は以下である。第 2 章では、定理証明支援系や大規模言語モデルの知識についてまとめる。第 3 章では、具体的な問題設定を行う。第 4 章では、本研究での手法や実験環境について説明する。第 5 章では、実験 I の実験についてまとめた。実験 I では、Coq の標準ライブラリを使用してデータセットを作成し、大規模言語モデルに追加学習を行たうえで tactic を生成させた。その結果、大規模言語モデルは何かしらの tactic を予測するのではなく、Coq のゴールを生成した。ただ大量のデータセットを用意し、一般的に用いられるプロンプトに当てはめて学習させる方法では効果が期待できないことがわかった。第 6 章では、実験 II の実験についてまとめた。実験 II では、自作のデータセットを作成し、大規模言語モデルに tactic とゴールを判別させた状態で学習できるか確認した。その結果、Coq のゴールに対して、最適な tactic を大規模言語モデルが出力するには LoRA で学習では変化せず、Full Fine-Tuning で調整する必要があるとわかった。第 7 章では、実験 III の実験についてまとめた。実験 III では学習データに含まれない定理の証明ができるかと学習の順番を変化させると結果が変わるのかについて検証した。その結果、本研究で作成したデータセットは形式言語を使用

した定理証明を目的とした大規模言語モデルへの追加学習に効果的であることがわかった。また、データを学習させる順番は学習にかかる時間が大きく変わり、性能にも影響がある可能性があることが分かった。第 8 章では、まとめとして考察と今後の課題、将来研究を議論する。付録 A には本研究のソースコードとその説明を添付した。

2 関連研究の概要

2.1 定理証明支援系

定理証明支援系 (**Proof Assistant**) とは、ステップバイステップでコンピュータに命令を送り、数学の定理を証明することを助けるシステムである [6]。定理証明支援系には Coq, AutoMath, Lean, Isabelle, Agda など様々ある。今回は Coq を用いる。

図 1 では、Coq IDE というソフトウェアを用いて Coq で証明を行っている途中の画面である。

定理証明支援系は検証だけではなく、定理証明を支援する機能がある。たとえば、証明の途中で残っているサブゴールを表示する機能である。Coq IDE 上では右上のブロックの内部にサブゴールが表示される。図 1 上では直線の下に書かれた P がサブゴールである。長い直線の上部には前提条件やすでに示されている定理が表示されている。

Coq のゴール 1 図 1 におけるゴールエリア

```

1 1 goal
2 ExMidLaw : forall P : Prop, P \ / ~P
3 P: Prop
4 HnotnotP : ~ ~ P
5 -----(1/1)
6 P

```

この右上のブロックをゴールエリアと呼び、表示されるものを Coq のゴールと呼ぶ。以降もこのゴールエリアに表示されるものを Coq のゴールと呼ぶ。

Coq にステップバイステップで命令を送り、Coq のゴールを遷移させ、サブゴールがなくなれば¹⁾、証明が終わったことを意味する。このステップバイステップで送る命令群を一般に **tactic** と呼ぶ。なお、本論文ではあるゴールに遷移するまでに実行された Coq のスクリプトを Coq の“入力スクリプト”と呼ぶ。具体的には、図 1 上で左のブロックに表示されているものが全体のスクリプトであり、その中でも緑色で塗られた部分はゴール 1 に遷移するまでの“入力スクリプト”である。

1) ゴールに No more goals と表示されるとサブゴールがないということである。

実際にゴールが遷移する様子を見る。ゴール 1 に `tactic -case:(ExMidLaw (~P)).` を実行すると Coq IDE 上では図 2 に変化し、ゴール 1 はゴール 2 に遷移する。

Coq のゴール 2 図 2 におけるゴールエリア

```

1 1 goal
2 ExMidLaw : forall P : Prop, P \ / ~P
3 P: Prop
4 HnotnotP : ~ ~ P
5 -----(1/2)
6 ~ P -> P
7 -----(2/2)
8 ~ ~ P -> P

```

この tactic では場合分けをしているため、サブゴールは二つになったため直線が二つになりそれぞれのサブゴールは表示されている。このように tactic を送り、ゴールを遷移させ、図 3 のように No more goals が表示されるまで続ける。これで証明完了である。

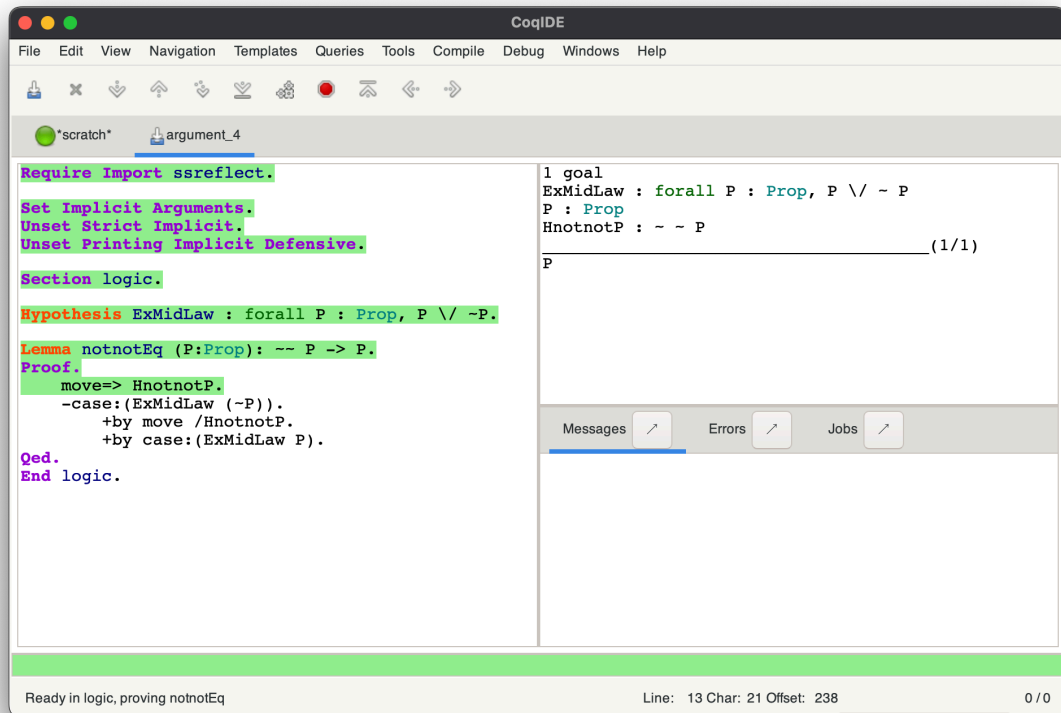


図 1 Coq IDE の表示画面

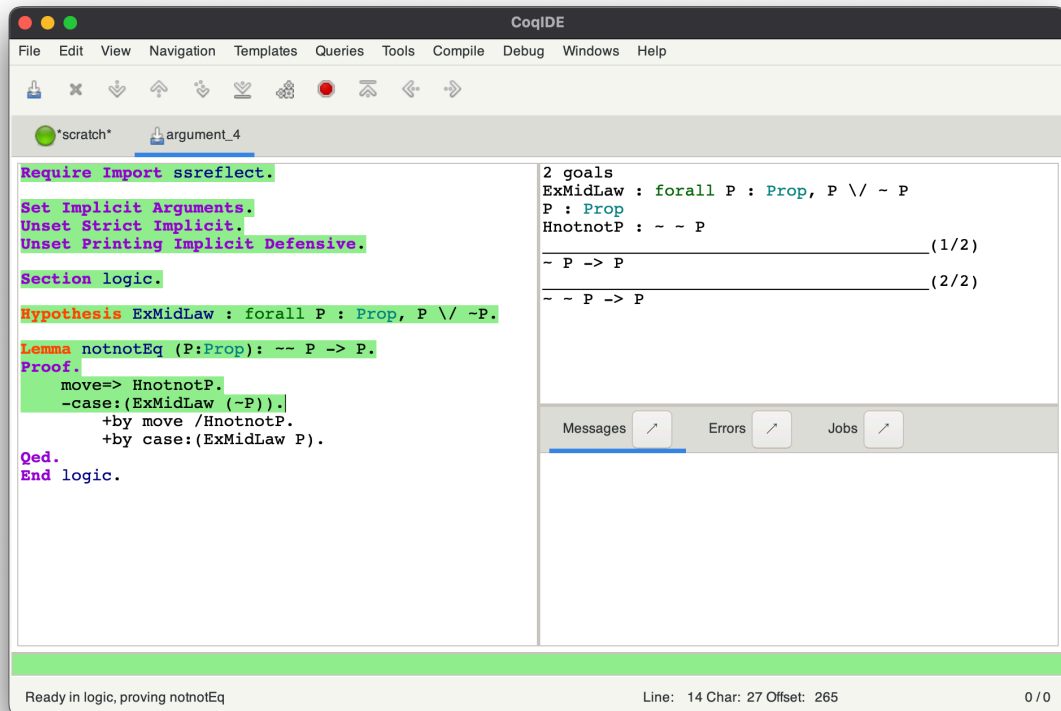


図 2 遷移後の Coq IDE の表示画面

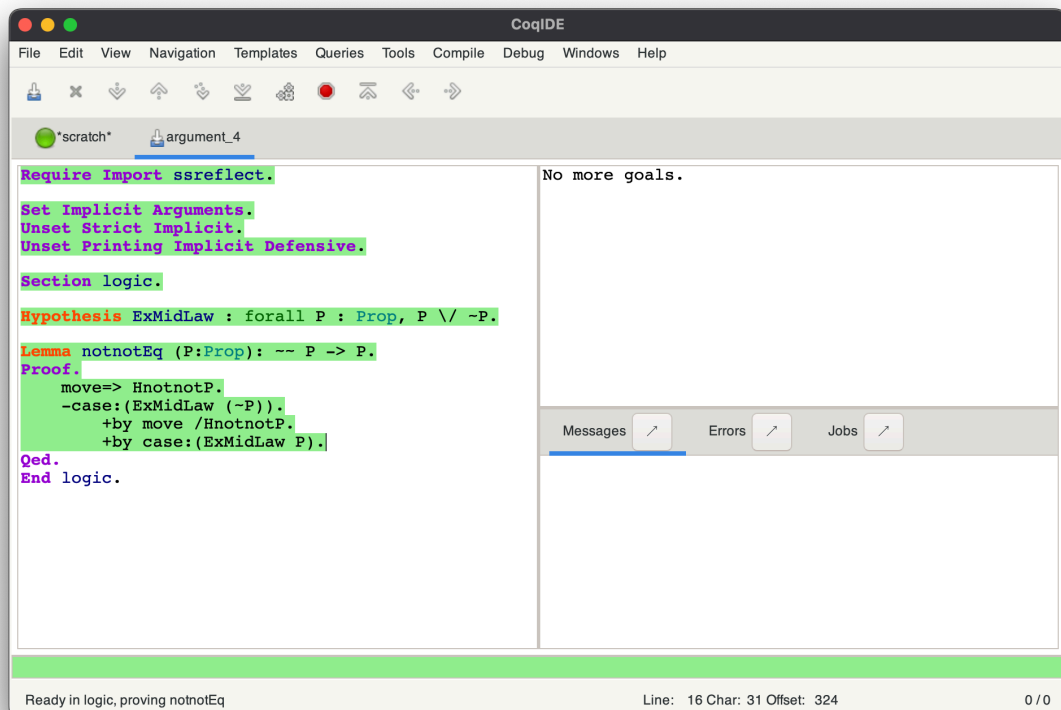


図 3 証明終了時の Coq IDE の表示画面

2.2 大規模言語モデル

大規模言語モデル (Large Language Models, LLM) とは、訓練された大規模なパラメータで構成されるニューラルネットワークである。通常、大規模なテキストデータを使って学習が行われる。

現在、コンピュータは、ASCII のように数値と対応させて文字を管理している。この数値は文字の意味を伴っていない。そのため、大規模言語モデルに読み込ませる際に意味情報を保ったまま数値にしたい。モデルに入力する際に **token** (トークン) という形に分解して数値ベクトルに変換する。ここでテキスト文を token に分割する処理を **tokenization** (トークン化) と呼ぶ。この分割した token を数値のベクトルに変換することでテキスト文を機械に読み込める形にする。このベクトルに変換したものを **Embedding** (埋め込み) と呼ぶ。現在大規模言語モデルで一般的に用いられる Embedding では、token どうしの関係性を表している。Embedding の性質として例えば、2 つの Embedding どうしの距離が近いならばその二つの token の関係性が高く、2 つの Embedding どうしの距離が遠いならばその二つの token の関係性が低いことを表している。

前段ではテキスト文を token に分けて機械に読み込ませる方法について紹介した。コンピュータに token を読み込ませることができたとしても token の並び、文章では文脈を読み取ることができなければテキスト文全体の理解をすることができない。そのため、文脈を含めた単語埋め込みである文脈化単語埋め込み (contextualized word embedding) を出力するモデルが開発された。また、機械翻訳のモデルとして Transformer(第 2.2.1 節で説明する) というモデルが提案された。

2.2.1 Transformer

Transformer [13] は、機械翻訳用のモデルとして提案されたが、言語生成や画像生成を含む様々なタスクに応用されている。2024 年 1 月の時点では Transformer をベースとしたモデルが標準的に利用されている。

図 4 の Transformer は大きく二つ、**Encoder** (エンコーダ) と **Decoder** (デコーダ) から構成されている。提案された時点では **Encoder-Decoder** (エンコーダ・デコーダ) の構成であったが、Encoder, Decoder それぞれのみの構成のモデルも利用されている。

Encoder (エンコーダ) と **Decoder** (デコーダ) の構造について整理する。Encoder は図 4 の左部, Decoder は図 4 の右部である。Encoder, Decoder について Encoder-Decoder で 1 token ずつ入力のテキスト文「私は猫を飼っている。」から翻訳した出力文「I have

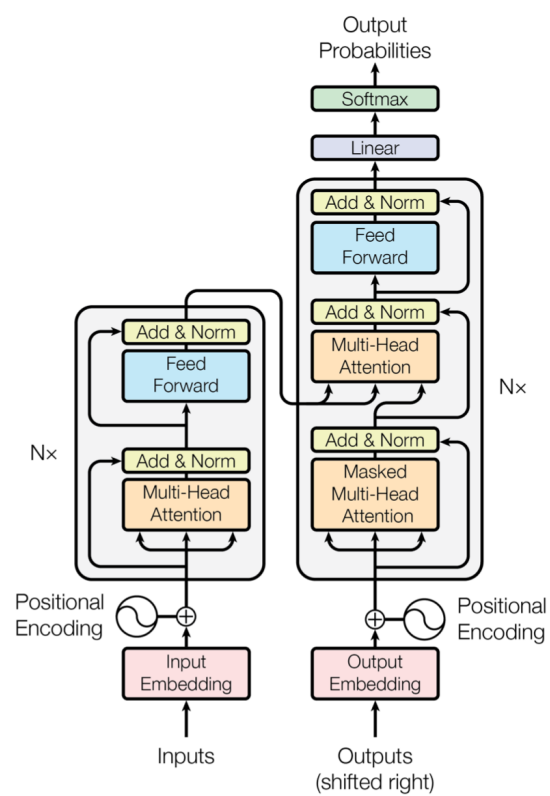


図 4 Transformer の全体図, Transformer の論文 [13] より引用

a cat.」を生成することから考える．Transformer は Encoder は，テキスト文「私は猫を飼っている。」を受け取り，原言語（この例では日本語）の文脈化単語埋め込みを出力する．Decoder は今までに生成した「I have a 」と Encoder で出力された文脈化単語埋め込みから次の単語である，「cat」を出力する．これを繰り返してテキスト文を生成していく．

次に，細かい処理を見る．inputs (tokenize した入力テキスト文) を input Embedding 層 (図 4 を参照) で token に対して D 次元の入力 token 埋め込みを出力する．そのため，出力されるのは token 数が V であれば， $V \times D$ 次元の入力埋め込み行列 $\mathbf{E} = [e_1, e_2, \dots, e_V]^T$ である．

次の Positional Encoding 層 (図 4 を参照) では，入力 token の埋め込みに対して，それぞれの token がどの位置に属するか **Positional Encoding** を与える．Positional Encoding は正弦関数を使ってベクトルで与える．具体的には， i 番目に位置する Positional Encoding を p_i とする．なお， p_i は D 次元である． p_i の j 番目の要素を $p_{i,j}$ とすると，

$$\begin{aligned} p_{i,2k+1} &= \sin\left(\frac{i}{10000^{2k/D}}\right) \\ p_{i,2k+2} &= \cos\left(\frac{i}{10000^{2k/D}}\right) \\ k &\in \left\{0, 1, \dots, \frac{D}{2} - 1\right\} \end{aligned}$$

そして，Positional Encoding からの出力は， i 番目の Positional Encoding からの出力を \mathbf{x}_i ，input Embedding 層の i 番目の出力を \mathbf{e}_i とすると，

$$\mathbf{x}_i = \sqrt{D}\mathbf{e}_i + p_{i,j}$$

とする．ここで \mathbf{e}_i を \sqrt{D} 倍しているのは Positional Encoding のスケールと揃えるためである．

次の Self-Attention 層 (図 4 の左では Multi-Head Attention 層²⁾) では，token の重要度を含めて文脈化を行う．ここでの処理は，それまで様々な処理をされて入力として与えられた Embedding を相互に参照して新しい Embedding を作成する．Transformer では query, key, value の三つの異なる Embedding を作成する．Self-Attention 層の出力を \mathbf{x} ，query, key, value をそれぞれ Q, K, V と表す．なお，Self-Attention 層の出力の次元は D である．

$$\mathbf{x} = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)V \quad (1)$$

2) Multi-Head Attention 層は，Self-Attention 層の処理を複数個並列で行い，最後に出力を一つにする．

softmax 関数は入力ベクトルの各成分を $[0, 1]$ の範囲の分布に変換する関数である．実際の処理ではまず，Self-Attention 層に入力された Embedding の各成分を線形変換して query, key, value ベクトルをそれぞれ生成する．次にある成分 \mathbf{e} の query ベクトルと各 token の key ベクトルの内積を取り，token 間の類似度を出す（式 1 で QK^T の部分）．ここで出た類似度を softmax で正規化し，合計 1 にする．ここで，値が大きければ大きいほど token 間の依存度は高い．この token 間の類似度と value を掛け，全ベクトルの総和を取る．この総和を \mathbf{e} の Self-Attention 層の出力とする．この操作をすべての成分の query ベクトルに対して行う．図 4 の右での Multi-Head Attention 層では query には Decoder の Embedding, key と value には Encoder の出力を用いる．入力以外で Multi-Head Attention 層とは違うところは最後の value の全ベクトルの総和を取るところでは重み付きで加算する．Masked Multi-Head Attention 層ではマスク処理が行われる．このマスク処理は次の単語を計算するごとにすべての計算をせず，高速化できるようにする処理である．

次の Feed Forward 層はサイズの大きい 2 層の Feed Forward neural network または multilayer perceptron; MLP と同じ構造になっている．Feed Forward 層の入力を \mathbf{x}_i とすると出力 \mathbf{y}_i は

$$\begin{aligned}\mathbf{y}_i &= \mathbf{W}_2 f(\mathbf{W}_1 \mathbf{x}_i + \mathbf{b}_1) + \mathbf{b}_2 \\ \mathbf{W}_1 &\in \mathbb{R}^{D_f \times D}, \mathbf{W}_2 \in \mathbb{R}^{D \times D_f} \\ \mathbf{b}_1 &\in \mathbb{R}^{D_f}, \mathbf{b}_2 \in \mathbb{R}^D\end{aligned}$$

で表すことができる．ここで $\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2$ は学習で調整するパラメータ， $f(\cdot)$ は活性化関数である．活性化関数には様々あり，提案時の Transformer では **ReLU(recified linear unit)** が用いられている．大規模言語モデルでは，**GELU(gaussian error linear unit)** が標準的に用いられている．

$$\text{ReLU}(x) = \max(0, x)$$

$$\text{GELU}(x) = x\Phi(x)$$

$\Phi(x)$ は標準正規分布 $N(0, 1)$ の累積分布関数である．

他の Add & Norm 層は学習を安定化させたり効率化するための層である．Add では残差接続を行っている．ひとつ前の層（Multi-Head Attention 層や Feed Forward 層）の入力と出力を足している．深層ニューラルネットワークの勾配消失・爆発問題を防ぐ．Norm では入力で与えられたベクトルを正規化している（レイヤー正規化；Layer

Normalization). Linear 層, Softmax 層は出力のための層であり, 次の token の生起確率を出力する.

これを大量のデータを用いて各層のパラメータを調整することでモデルの精度を上げる.

2.2.2 Fine-Tuning

特定のタスクを解く前に, 別のタスクで訓練することを**事前学習 (pre-training)**と呼ぶ. 事前学習済みのモデルのパラメータ調整方法には **Fine-Tuning** がある. Fine-Tuning は, 新しいデータセットを用いて新たに加えた層と事前学習モデル全体に調整を与えることである. 一方, 追加で学習をせずにテキスト文で命令や質問を工夫することで大規模言語モデルに目的に合った出力をさせたり, 出力の精度を向上させることでタスクを解く方法がある. この言語モデルに対して人間が入力する命令や質問などのテキスト文は, プロンプトとよばれる.

全てのパラメータに対して Fine-Tuning することを **Full Fine-tuning** と呼ぶ. **PEFT (Parameter-Efficient Fine Tuning)** は, 一部のパラメータのみの調整で Full Fine-Tuning と同等の性能が期待できる手法であり, 計算コストを削減することができる.

2.2.3 LoRA

LoRA (Low-Rank Adaptation) は, PEFT (Parameter-Efficient Fine Tuning) の手法のひとつである. LoRA は, 事前学習済みモデルの Attention 層に対してサイズの小さい行列を適用することで更新するパラメータ数を削減しながら, 推論時の速度を Fine Tuning した場合と同等に保つことが出来る手法である. 事前学習済みモデルのある Linear 層のパラメータを $\mathbf{W} \in \mathbb{R}^{D_{in} \times D_{out}}$, 入力ベクトルを $\mathbf{x} \in \mathbb{R}^{D_{in}}$ とすると, 出力 $\mathbf{y} \in \mathbb{R}^{D_{out}}$ は

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

である. Fine-Tuning はこの \mathbf{W} を直接調節する. LoRA ではこの \mathbf{W} を固定し, 新しい差分 $\Delta\mathbf{W}$ を導入し, この差分を調整する. この時 LoRA での出力は $\mathbf{y} \in \mathbb{R}^{D_{out}}$ は

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \Delta\mathbf{W}\mathbf{x}$$

と計算する. LoRA で導入した差分 $\Delta\mathbf{W}$ を二つの行列に分解することで学習するパラメータを減らす. これにより, 学習に必要なメモリの削減, 必要な学習時間を減らすこと

ができる。

$$\Delta \mathbf{W} = \mathbf{A}\mathbf{B}$$

$$\mathbf{A} \in \mathbb{R}^{D_{in} \times r}, \mathbf{B} \in \mathbb{R}^{r \times D_{out}}$$

ここで、学習するパラメータは \mathbf{A}, \mathbf{B} の要素である。ここで、 r のサイズを変更することで学習するパラメータの数を調整することができる。追加学習の初期では \mathbf{A} がランダムなパラメータ、 $\mathbf{B} = \mathbf{0}$ とし、 $\Delta \mathbf{W} = \mathbf{0}$ と差分導入前と一致させる。

これを図に表すと図 5 である。

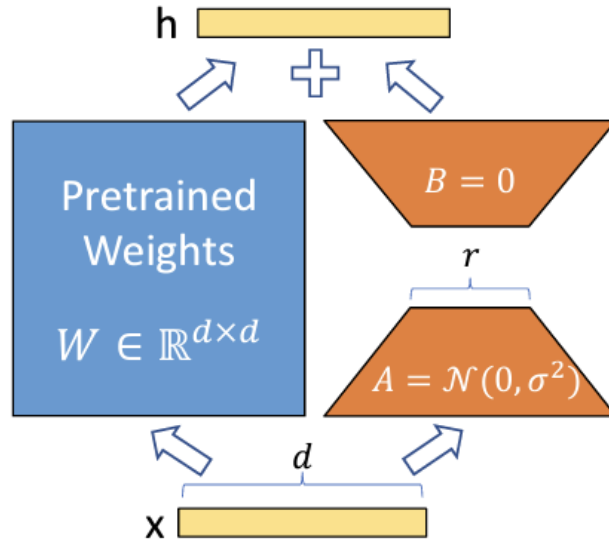


図 5 LoRA の論文 [7] Fig.1 より引用

3 問題設定

本研究では、Coq のゴールに対して、最適な tactic を大規模言語モデルが出力できるか検証する。具体的には、Coq の状態であるゴール情報から言語モデルに Coq の次の tactic を提案させる。

前提として、今回 Coq によって証明する定理は真で、証明ができることが判明しているものとする。また、言語モデルに与える Coq の“入力スクリプト”は Coq で実行した際にエラーが出力されないものとする。

今回の問題では、Coq のゴールを表すテキスト情報を言語モデルに入力として与える。そして Coq のゴールを表すテキスト情報に対して予想される Coq の次の tactic (Coq のスクリプト) を言語モデルに出力させる。この問題では、Coq の“入力スクリプト”を実行した後の Coq のゴールを表すテキストから言語モデルが予測した Coq の次の tactic により Coq のゴールが進む、つまり証明が進むことが望ましい。

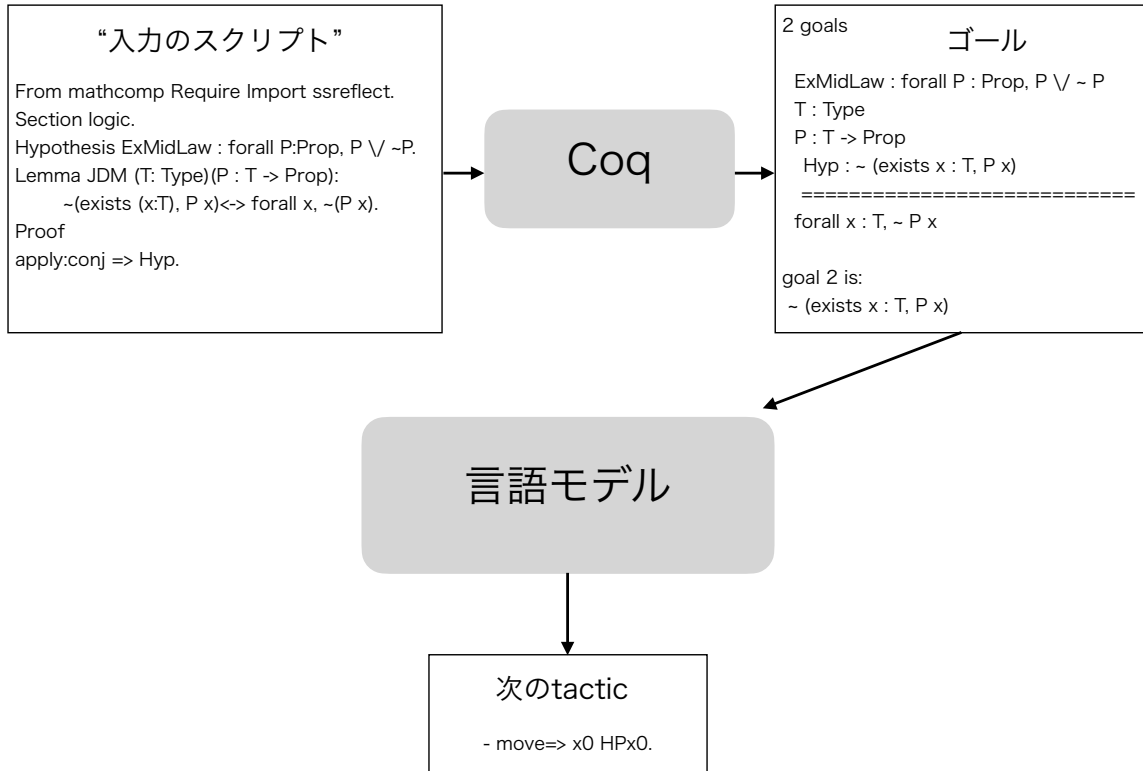


図 6 Coq と大規模言語モデルによる学習の例

3.1 相互学習

Coq のスクリプトと Coq のゴールを表すテキスト情報から大規模言語モデルを学習させ、大規模言語モデルに Coq のスクリプトを予測させることによって Coq による証明を進めることを Coq と大規模言語モデルによる相互学習とする。

4 手法

本研究では，Coq の“入力スクリプト”と Coq ゴールから大規模言語モデルに Coq の次の tactic を提案させる．

本研究での提案手法の構造を図 7 で示す．

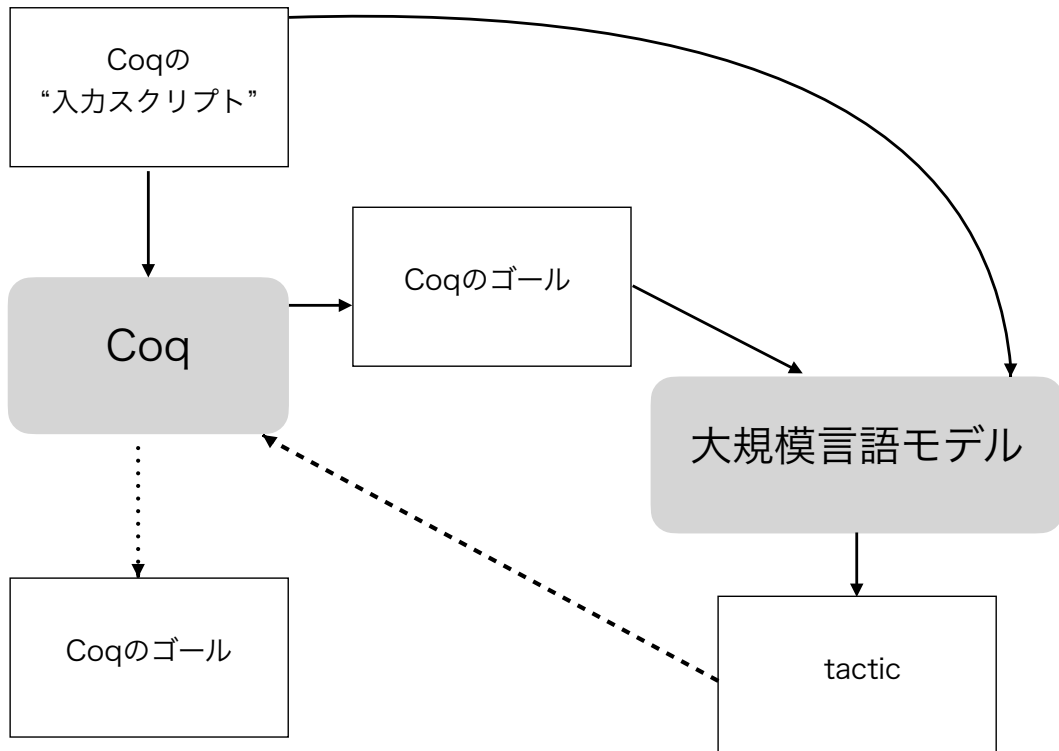


図 7 提案手法の構造

既存の大規模言語モデルに関する研究では，自然言語によるテキスト文やプログラムなど単一の要素のみを対象とした研究が盛んである．本研究では，Coq の“入力スクリプト”と Coq のゴールと二つの要素で構成されているような構造が違うものに対してどう適用させるかを提案する．

Lean や Agda などの対話システムの定理証明支援系に対しても同様に言語モデルとの相互学習が適用できる．今回の提案手法では定理証明支援系へ命令するスクリプトと定理証明支援系の状態から次の定理証明支援系へ命令するプログラムを予想する方法であるため，他の対話システムに適用可能である．さらに，スクリプトと状態に着目すると，スク

リプトと状態の変化はそれぞれ違う変化をする．強化学習のように行動と状態という違う二つの関連性のある項目をまとめ，学習させる良い方法の提案となる．

大規模言語モデルに数学の問題を解かせる際に自然言語をそのまま学習に使用している．本研究の提案手法は，数学の問題（定理証明）を形式化することで自然言語より大規模言語モデル学習しやすいデータ構造で数学を解かせている．

Coq の tactic 予想に関する既存研究では，ルールベースのものや初期のニューラルネットワーク等を使用している．今回の提案手法は大規模言語モデルを使用している．

定理証明支援系 Lean と大規模言語モデルの研究である，Lean-dojo[15] では，Lean のゴールとそれに対する tactic の組み合わせのみを tactic 予測のために使用している．今回の提案手法では，Coq のゴールとそれに対する tactic だけではなく，それ以前の Coq の“入力スクリプト”を情報として加えている．

4.1 実験環境

本研究に用いた環境は以下である．

ハードウェア構成は，CPU が Intel(R) Xeon(R) w5-2455X (ベースクロック 3.2Ghz, 12 コア) 1 枚，メモリが 64GB (16GBx4) DDR5-4800 ECC Registered, GPU が NVIDIA RTX A6000 (48 GB GDDR6) 2 枚である．

ソフトウェア構成は，OS が Ubuntu 22.04.3 LTS である．cuda のバージョンは 12.1.105 である．Python のバージョンは 3.10.12, transformers は 4.35.0.dev0, PyTorch が 2.0.1 である．OCaml のバージョンは 4.14.0, Coq のバージョンは 8.18.0, MathComp のバージョンは 2.1.0 である．

4.2 パラメータ

各実験の設定にてパラメータの値を表記している．ここでパラメータについて整理する．

- base model

使用した事前学習済みモデルを指定している．

- 学習手法

モデルの再学習でどの手法を使っているかを指定している．例えば Fine-Tuning や LoRA である．

- epoch

どの学習方法でも一つの訓練データを繰り返して学習させる．このパラメータはそ

の繰り返し回数を指定する.

- batch

学習するデータセットを一度に学習するのではなく, 学習するデータセットをいくつかのグループ (batch) に分けて学習する. このグループに分けるときの何個ずつに分けるか batch size で指定する. ここでは per_device_train_batch_size で指定する. 学習中, 1GPU ごと割り振る batch size である.

- 学習率

学習の最適化でパラメータを一度にどの程度変化させるかを表す.

- optim

このパラメータは最適化アルゴリズムを指定する.

- lr_scheduler_type

学習率を変化させることで学習を効率的に行う手法がある. このパラメータはその種類を指定している. 今回は "constant" (設定した一定値の学習率で学習を行う) を指定している.

- max_seq_length

トークン化した後の最大合計入力シーケンス長を指定する. 指定した値より長いシーケンスは切り捨てられ, 短いシーケンスはパディングされる. None を指定した際は特に変更を加えない.

- lora_r

このパラメータは第 2.2.3 目 LoRA にて差分 ΔW を二つの行列に分解する際に指定する行列のサイズ r を指定する.

- lora_alpha

LoRA 学習において出力 $\mathbf{y} = \mathbf{W}\mathbf{x} + \Delta\mathbf{W}\mathbf{x}$ を出す際に差分 ΔW に値 lora_alpha / lora_r を掛ける. これは差分 ΔW における学習率のようなものである. 今回, lora_alpha は lora_r の二倍の値で指定した.

- lora_dropout

学習の途中に一定確率で, 特定のパラメータをドロップする (値を 0 にする). この確率を指定する. これにより特定の層への偏りを減らし, 性能向上につながる.

- target_modules

どの層で LoRA 学習をするのか指定する.

5 実験 I

この章では、Coq の標準ライブラリである、theories ライブラリを使用した実験について整理する。

5.1 データセット

データセットは新規構築した。例に出すデータセットの例は視認性のために適宜改行を加えている。Coq の標準ライブラリである、theories ライブラリ [5] の Coq のソースファイルからデータセットを作成した。theories ライブラリに含まれている Coq のスクリプトファイルを n 行目まで Coq で実行し、Coq のゴールが返ってきたものをデータセットとして使用した。具体的なデータの作成方法は付録 A.2 に記す。データの構成は、Alpaca dataset[12] をもとに作成した。データフィールドは、

- instruction (str) : Coq に入力した n 行目までの Coq のスクリプト
- input(str): Coq のゴール
- output(str): 次の tactic (Coq スクリプトの $n+1$ 行目)

とした。また、instruction のテキスト文のなかですでに証明済みで証明が書かれているもの (Proof. から始まり Qed. で閉じているものは自明とし、Admitted. へ変換した)。実際のデータの具体例 1 を下記に表記する。

データの具体例 1 theories ライブラリから作成したデータの例

```
1 {
2 'instruction':
3 "(*****\n
4 (*      * The Coq Proof Assistant / The Coq Development Team      *)\n
5 (* v      * Copyright INRIA, CNRS and contributors                  *)\n
6 (* <0_--,, * (see version control and CREDITS file for authors & dates) *)\n
7 (*  \\VV/  *****\n
8 (*  //    * This file is distributed under the terms of the        *)\n
9 (*      * GNU Lesser General Public License Version 2.1            *)\n
10 (*      * (see LICENSE file for the text of the license)            *)\n
11 (*****)\n
12 \n
13 Require Import NAXioms NSub NDiv.\n
14 \n
15 Module Type NDivPropPrivate (N : NAXiomsSig') (NP : NSubProp N).\n
16 Declare Module Private_NDivProp : NDivProp N NP.\n
17 End NDivPropPrivate.\n
18 \n
19 (** Properties of Euclidean Division with  $a / 0 == 0$  and  $a \bmod 0 == a$  *)\n
20 \n
21 Module Type NDivProp0\n
22 (Import N : NAXiomsSig')\n
23 (Import NP : NSubProp N) (Import D0 : NZDivSpec0 N N N)\n
24 (Import P : NDivPropPrivate N NP).
```

```

25 \n
26 Import Private_NDivProp.\n
27 \n
28 (** Let's now state again theorems, but without useless hypothesis. *)\n
29 \n
30 Module Div0.\n
31 \n
32 Lemma div_0_l : forall a, 0/a == 0.\n
33 Admitted.\n
34 \n
35 Lemma mod_0_l : forall a, 0 mod a == 0.\n
36 Admitted.\n
37 \n
38 #[local] Hint Rewrite div_0_l mod_0_l div_0_r mod_0_r : nz.\n
39 \n
40 Lemma div_mod : forall a b, a == b*(a/b) + (a mod b).\nAdmitted.\n
41 \n
42 Lemma mod_eq : forall a b, a mod b == a - b*(a/b).\n
43 Admitted.\n
44 \n
45 Lemma mod_same : forall a, a mod a == 0.\n
46 Admitted.\n
47 \n
48 Lemma mod_mul : forall a b, (a*b) mod b == 0.\n
49 Admitted.\n
50 \n
51 Lemma mod_le : forall a b, a mod b <= a.\n
52 Admitted.\n
53 \n
54 Lemma div_le_mono : forall a b c, a<=b -> a/c <= b/c.\n
55 Admitted.\n
56 \n
57 Lemma mul_div_le : forall a b, b*(a/b) <= a.\n
58 Admitted.\n
59 \n
60 Lemma div_exact : forall a b, (a == b*(a/b) <-> a mod b == 0).\n
61 Admitted.\n
62 \n
63 Lemma div_lt_upper_bound : forall a b q, a < b*q -> a/b < q.\n
64 Admitted.\n
65 \n
66 Lemma div_le_upper_bound : forall a b q, a <= b*q -> a/b <= q.\n
67 Admitted.\n
68 \n
69 Lemma mod_add : forall a b c, (a + b * c) mod c == a mod c.\n
70 Admitted.\n
71 \n
72 Lemma div_mul_cancel_r : forall a b c, c~=0 -> (a*c)/(b*c) == a/b.\n
73 Proof.\n
74 intros a b c. destruct (eq_decidable b 0) as [->|Hb].\n
75 - now nzsimpl.\n",
76 'input':
77 '100a101,107\n
78 > This subproof is complete, but there are some unfocused goals.\n
79 > Focus next goal with bullet -.\n> \n> 1 goal\n
80 > \n
81 > goal 1 is:\n
82 > c ~= 0 -> a * c / (b * c) == a / b\n',
83 'output': ' - now apply div_mul_cancel_r.\n'}

```

theories ライブラリから作成したデータセットは、Alpaca dataset で学習させる際に標準で用いられるプロンプトで大規模言語モデルに学習させた。下記にプロンプト 1 を表記する。なお、例中の{instruction}, {input_text}, {response}にはそれぞれデータフィールドの instruction, input_text, response を入力するものとする。

プロンプト 1 学習時に使用したプロンプト

```
1 f'''Below is an instruction that describes a task, paired with an input that
  provides further context. Write a response that appropriately completes the
  request.
2
3     ### Instruction:
4     {instruction}
5
6     ### Input:
7     {input_text}
8
9     ### Response:
10    {response}
11    '''
```

データセットのファイルサイズは 871 MB である。データのサンプル数は、84,723 である。各データの文字数の平均は、instruction, input, output がそれぞれ 9,459, 350, 31 であり、プロンプトで加工したあと、つまり入力時の文字数の平均は 10,117 である。

5.2 実験設定

学習時の設定を以下の表 1 にまとめた。学習に要した時間は、約 39 時間である。

表 1 学習の設定

学習手法	LoRA
base model	meta-llama/Llama-2-7b-hf
epoch	1
batch	4
学習率	2e-5
optim	paged_adamw_32bit
lr_scheduler_type	"constant"
max_seq_length	None
lora_r	64
lora_alpha	128
lora_dropout	0.1
target_modules	q_proj, k_proj, v_proj, o_proj

5.3 実験結果

大規模言語モデルに推論させ、テキスト文を生成させた例 2 と例 4 を記す。なお、大規模言語モデルに指定した出力のトークン数は、入力された文のトークン数 +100 とした。

例に用いたデータはデータの具体例 1, プロンプトは学習に用いたものプロンプト 1 と同じものとし, ### Response: で区切り, それ以降を推論させる. 推論に要した時間は 7 秒である. 入力したテキストはデータの具体例 2, それに対する出力してほしいテキストはデータの具体例 3 である.

データの具体例 2 入力したテキスト

```

1 "Below is an instruction that describes a task, paired with an input that provides
  further context. Write a response that appropriately completes the request.\n
2 \n
3 "### Instruction:\n
4 (*****\n
5 (*      * The Coq Proof Assistant / The Coq Development Team      *)\n
6 (* v      *      Copyright INRIA, CNRS and contributors      *)\n
7 (* <0_--,, * (see version control and CREDITS file for authors & dates) *)\n
8 (* \\VV/   *****\n
9 (* // *      This file is distributed under the terms of the      *)\n
10 (*      *      GNU Lesser General Public License Version 2.1      *)\n
11 (*      *      (see LICENSE file for the text of the license)      *)\n
12 (*****\n
13 \n
14 Require Import NArith NSub NDiv.\n
15 \n
16 Module Type NDivPropPrivate (N : NArithSig) (NP : NSubProp N).\n
17 Declare Module Private_NDivProp : NDivProp N NP.\n
18 End NDivPropPrivate.\n
19 \n
20 (** Properties of Euclidean Division with a / 0 == 0 and a mod 0 == a *)\n
21 \n
22 Module Type NDivProp0\n
23 (Import N : NArithSig)\n
24 (Import NP : NSubProp N)\n
25 (Import D0 : NZDivSpec0 N N N)\n
26 (Import P : NDivPropPrivate N NP).\n
27 \n
28 Import Private_NDivProp.\n
29 \n
30 (** Let's now state again theorems, but without useless hypothesis. *)\n
31 \n
32 Module Div0.\n
33 \n
34 Lemma div_0_l : forall a, 0/a == 0.\n
35 Admitted.\n
36 \n
37 Lemma mod_0_l : forall a, 0 mod a == 0.\n
38 Admitted.\n
39 \n
40 #[local] Hint Rewrite div_0_l mod_0_l div_0_r mod_0_r : nz.\n
41 \n
42 Lemma div_mod : forall a b, a == b*(a/b) + (a mod b).\n
43 Admitted.\n
44 \n
45 Lemma mod_eq : forall a b, a mod b == a - b*(a/b).\n
46 Admitted.\n
47 \n
48 Lemma mod_same : forall a, a mod a == 0.\n
49 Admitted.\n
50 \n
51 Lemma mod_mul : forall a b, (a*b) mod b == 0.\n
52 Admitted.\n
53 \n
54 Lemma mod_le : forall a b, a mod b <= a.\n
55 Admitted.\n
56 \n
57 Lemma div_le_mono : forall a b c, a<=b -> a/c <= b/c.\n

```

```

58 Admitted.\n
59 \n
60 Lemma mul_div_le : forall a b, b*(a/b) <= a.\n
61 Admitted.\n
62 \n
63 Lemma div_exact : forall a b, (a == b*(a/b) <-> a mod b == 0).\n
64 Admitted.\n
65 \n
66 Lemma div_lt_upper_bound : forall a b q, a < b*q -> a/b < q.\n
67 Admitted.\n
68 \n
69 Lemma div_le_upper_bound : forall a b q, a <= b*q -> a/b <= q.\n
70 Admitted.\n
71 \n
72 Lemma mod_add : forall a b c, (a + b * c) mod c == a mod c.\n
73 Admitted.\n
74 \n
75 Lemma div_mul_cancel_r : forall a b c, c~0 -> (a*c)/(b*c) == a/b.\n
76 Proof.\n
77 intros a b c. destruct (eq_decidable b 0) as [->|Hb].\n
78 - now nzsimpl.\n
79 \n
80 \n
81 ### Input:\n
82 100a101,107\n
83 > This subproof is complete, but there are some unfocused goals.\n
84 > Focus next goal with bullet -.\n
85 > \n
86 > 1 goal\n
87 > \n
88 > goal 1 is:\n
89 > c ~0 -> a * c / (b * c) == a / b\n
90 \n
91 \n
92 ### Response:\n "

```

期待される出力は以下のデータの具体例 3 である。

データの具体例 3 期待する答え

```

92 ### Response:\n
93 - now apply div_mul_cancel_r.\n"

```

実際に出力したのは以下のデータの具体例 4 である。

データの具体例 4 推論した部分

```

92 ### Response:\n
93 100a101,107\n
94 > This subproof is complete, but there are some unfocused goals.\n
95 > Focus next goal with bullet -.\n
96 > \n
97 > 1 goal\n
98 > \n
99 > goal 1 is:\n
100 > a ~0 -> a / b == a / (b * c) / c\n
101 \n
102 \n
103 ### Instruction:\n
104 100a101,107\n"

```

このようにデータの具体例 3 のような答えを出さずに、Coq のゴールを生成している。

プロンプトを工夫して### Response が Coq のスクリプト、### Input が Coq のゴー

ルであり，次の tactic を予想するように明記しても Coq のゴールを予想し，データの具体例 4 と変わらなかった．

5.4 考察

間違えた tactic を予測するのではなく，Coq のゴールを生成した．そのため，大規模言語モデルには Coq の tactic と Coq のゴールを判別できていないと考えられる．この原因として大きく二つ考えられる．一つ目は，全体の文が長いこと，特に `### Instruction` が長いため，`### output` の情報量が減ってしまったことで学習がうまくいかなかったことが考えられる．二つ目は，データセットに用いた Coq のスクリプトである．このデータセットに用いた Coq のスクリプトは基本となるデータセットであり，ソースの半分以上を定義が占めている．証明をしている部分も，自明である³⁾とし，Coq の検証で証明が終わっているものや，工夫をしている証明があったとしても 1,2 行で終わっており，証明全体を考慮して tactic を工夫する証明が少ない．

以上のように，この実験では，ただ大量のデータセットを用意し，一般的に用いられるプロンプトに当てはめて学習させる方法では効果が期待できないことがわかった．

6 実験 II

この実験では，大規模言語モデルに tactic とゴールを判別させた状態で学習できるか確認した．

6.1 データセット

第 5 章実験 I の結果を元に，データセットは新規構築した．具体的なデータの作成方法は付録 A.3 に記す．現在 Hugging Face [1] 等で公開されている公開データセットには一部 Coq のスクリプトが含まれているが，そのスクリプトを Coq で実行した際に Coq から返ってくる返答はほとんど含まれておらず情報量が足りていないため使用しなかった．

新しく構築したデータセットは二種類である．例に出すデータセットの例は視認性のために適宜改行を加えている．

一つ目のデータセットは元データとなる Coq のスクリプトを作成することから始めた．作成した Coq のスクリプトは以下の定理とその類似の定理を Coq で証明したものである．

3) tactic では `by[]`. と入力する

- 命題論理

- $P \leftrightarrow \neg\neg P$
- $(A \rightarrow B) \rightarrow (\neg B \rightarrow \neg A)$
- $\neg(A \wedge B) \leftrightarrow \neg A \vee \neg B$
- $(A \rightarrow B) \wedge (A) \rightarrow (B)$

- 一階述語論理

- $\neg(\forall x, P(x)) \leftrightarrow \exists x, \neg(P(x))$

- ペアノ算術

- $n \in \mathbb{N}, 0 + n = n$

データの構成は、ChatGPT API の role [10] を参考に対話形式のものにした。作成したデータのデータフィールドは "text"(str) のみとした。"text" の内部は

- **##System** : Coq のスクリプトで最初に表記する定義や証明する定理等
- **##Assistant** : Coq から返ってきた Coq のゴール (**##User** に表記したもの) から Coq へ入力した tactic
- **##User** : **##System** や **##Assistant** に書かれた Coq の"入力スクリプト"を Coq へ入力した際に Coq から返ってきた Coq のゴール

とし、**##System** の情報から始め、**##Assistant** (tactic) と **##User** (Coq のゴール) の対話形式とした。これ以降、この作成したデータセットを対話形式のデータセットと呼ぶ。データセットのファイルサイズは 55.7 KB である。データのサンプル数は、24 である。入力の単語数の平均は、2,148 である。実際のデータの具体例 5 を下記に表記する。

データの具体例 5 対話形式のデータセット

```

1 {
2   "text": "##System\n
3   From mathcomp Require Import ssreflect ssrnat.\n
4   Section naturalNumber.\n
5   Lemma addOnEqn (n:nat) : 0+n=n.\n
6   Proof.\n
7   ##User\n
8   1 goal\n
9   \n
10  n : nat\n
11  =====\n
12  0 + n = n\n
13  ##Assistant\n
14  by [].\n
15  ##User\n
16  No more goals.\n
17  ##Assistant\n
18  Qed.\n
19  End naturalNumber.\n"
20 }
```

二つ目のデータセットは一つ目で作成したデータセットを使用し、選択問題形式のデータセットに加工したものである。具体的には、`##Assistant` を選択問題形式に変更した。二つ目のデータセットでは

データの具体例 6 対話形式のデータセットの **Assistant** 部分

```
1 ##Assistant\n
2 by [].\n"
```

のように tactic のみ表記していたものを

データの具体例 7 選択問題形式のデータセットの **Assistant** 部分

```
1 ##Assistant\n
2 ``Select the best tactic as the next instruction to be given to Coq from the following
   .\n
3 The choices are\n
4 1. by [].\n
5 2. by apply :n.\n
6 3. rewrite mulnC.\n
7 Answer choice number is:1``\n
8 by [].\n"
```

のように選択問題とその答えを作成し、“”で囲み、次の行に実際の tactic を表記した。選択問題の選択肢は自作し、正解に近い選択肢だけでなく、スペルミスのあるものや tactic がない一般的な名詞 (apple 等) とした。選択肢の数は固定せず 3 つのものや 6 つのものなどランダムに作成した。なお `##User` の部分は選択問題の部分のみにし、入力の長さを減らした。これ以降、この作成したデータセットを選択問題形式のデータセットと呼ぶ。データセットのファイルサイズは解答を含めて 181 KB である。データのサンプル数は、188 である。入力の単語数の平均は、909 である。なお、選択肢の数が 3 つのものが 103 問、4 つのものが 74 問、5 つのものが 8 問、6 つのものが 3 問である。実際のデータの具体例 8 を下記に表記する。

データの具体例 8 選択問題形式のデータセット

```
1 {
2   "text": "##System\n
3   From mathcomp Require Import ssreflect ssrnat.\n
4   Section naturalNumber.\n
5   Lemma add0nEqn (n:nat) : 0+n=n.\n
6   Proof.\n
7   ##User\n
8   1 goal\n
9   \n
10  n : nat\n
11  =====\n
12  0 + n = n\n
13  ##Assistant\n
14  ``Select the best tactic as the next instruction to be given to Coq from the
    following.\n
15  The choices are\n
16  1. by [].\n
17  2. by apply :n.\n
18  3. rewrite mulnC.\n
```



```

19   Answer choice number is:1```\n
20   by[].\n"
21 }

```

6.2 実験設定

訓練集合と検証集合，テスト集合と分割せずに学習を行い，訓練に使用したデータでそのまま検証した．大規模言語モデルが Coq の tactic の形式を認識することができるか検証するためにこの方式で行った．また，今回の大規模言語モデルに対する学習では，特にプロンプトを指定せず，対話形式のデータセットと選択問題形式のデータセットはそのまま一つのテキスト文として入力した．

大規模言語モデルに，入力するテキスト文は選択肢を選ぶ直前 (Answer choice number is :) まで入力し，大規模言語モデルにその続きに最適な選択肢を出力することを期待する．なお，大規模言語モデルに指定した出力のトークン数は，入力された文のトークン数 + 100 とした．以下に入力 9 と期待される出力 10 を示す．

データの具体例 9 テストのために入力されるテキスト文

```

1  "##System\n
2  From mathcomp Require Import ssreflect ssrnat.\n
3  Section naturalNumber.\n
4  Lemma addOnEqn (n:nat) : 0+n=n.\n
5  Proof.\n
6  ##User\n
7  1 goal\n
8  \n
9  n : nat\n
10 =====\n
11 0 + n = n\n
12 ##Assistant\n
13 ``Select the best tactic as the next instruction to be given to Coq from the following\n
14   .\n
15 The choices are\n
16 1. by[].\n
17 2. by apply :n.\n
18 3. rewrite mulnC.\n
19 Answer choice number is:

```

期待される出力は，以下のデータの具体例 10 である．省略している最初の部分は入力の文 9 と同じ文である．

データの具体例 10 期待される出力

```

17 ...
18 Answer choice number is:1```\n
19 by[].\n"

```

各学習時の設定，学習に要した時間を以下の表 2 にまとめた．

表 2 学習の設定

model 名	model L1	model L2	model F1
使用した データセット	対話形式	対話形式 選択問題形式	対話形式
学習手法	LoRA	LoRA	Full Fine-Tuning
base model	meta-llama/ Llama-2-7b-hf	meta-llama/ Llama-2-7b-hf	meta-llama/ Llama-2-7b-hf
epoch	10	10	10
batch	4	4	2
学習率	2e-5	2e-5	2e-5
optim	paged_ adamw_32bit	paged_ adamw_32bit	paged_ adamw_32bit
lr_ scheduler_type	"constant"	"constant"	"constant"
max_seq_length	None	None	None
lora_r	64	64	
lora_alpha	128	128	
lora_dropout	0.1	0.1	
target_modules	q_proj, k_proj, v_proj, o_proj	q_proj, k_proj, v_proj, o_proj	
学習に要した 時間	39 sec	14 min	44 min

6.3 検証方法

大規模言語モデルに選択問題を解かせ、その正解率を評価点とする。ここでは大規模言語モデルが選択問題を完答できるか評価する。大規模言語モデルが正解の選択肢を選んだら 1、それ以外の選択肢を出力した場合や選択肢を出力しない場合は 0 とする。この評価点の平均を取り、その値を評価指標の数値とする。最大値は 1 である。

学習させた内容が反映されているか確認するために学習させた問題を全てそのまま評価した。

条件なくランダムに選択肢よりも精度が高いかどうか検証するためにランダムに選択肢を選んだ場合の確率の割合とスコア比較をする。具体的には、ランダムに選択肢を選んだ場合、選択肢が n 個ならば $1/n$ の確率で正解する。その割合と比較する。今回は選択肢の個数は固定していないので、それぞれのランダムに選択肢を選んだ場合の確率の平均をとる。

検証では、ランダムに選択肢を選んだ場合の確率の割合と元モデルとのスコア比較をする。

6.4 実験結果

ランダムに選択肢を選んだ場合の確率の割合と元モデルのスコアを表 3 にまとめた。

表 3 基準のスコア

	ランダムに選択肢を選んだ 場合の確率の割合	meta-llama/ Llama-2-7b-hf
推論に要した 平均時間		4 sec
score	0.29	0.13
問題数	188	188
正解数	55	25
不正解数		69
数字を 生成しなかった数		94

各学習時の設定ごとの推論結果の各スコアを表 4 にまとめた。スコアは四捨五入して小数点第二位までとした。

表 4 実験 II 結果

modele 名	model L1	model L2	model F1
使用した データセット	対話形式	対話形式 選択問題形式	対話形式
学習手法	LoRA	LoRA	Full Fine-Tuning
base model	meta-llama/ Llama-2-7b-hf	meta-llama/ Llama-2-7b-hf	meta-llama/ Llama-2-7b-hf
推論に要した 平均時間	4 sec	4 sec	4 sec
score	0.10	0.10	0.30
問題数	188	188	188
正解数	18	18	57
不正解数	57	57	111
数字を 生成しなかった数	113	113	20

以下に実際の生成例を記載する.

まずは大規模言語モデルが正解を生成した例 11 である.

データの具体例 11 大規模言語モデルが正解を生成した際の出力例

```

1  """##System\\n
2  Require Import ssreflect.\\n
3  Section HilbertSAxion.\\n
4  Variable A B C : Prop.\\n
5  Theorem HS1 : ( A -> ( B -> C ) ) -> ( ( A -> B ) ->(A-> C ) ).\\n
6  Proof.\\n
7  ##User\\n
8  1 goal\\n
9  \\n
10 A, B, C : Prop\\n
11 =====\\n
12 (A -> B -> C) -> (A -> B) -> A -> C\\n
13 ##Assistant\\n
14 ``Select the best tactic as the next instruction to be given to Coq from the following
15 .\\n
16 The choices are\\n
17 1. rewrite /iff.\\n
18 2. move=> A_to_B_to_C_is_true.\\n
19 3. case.\\n
20 Answer choice number is:'2'.\\n
21 ##Assistant\\n
22 move=> A_to_B_to_C_is_true.\\n
23 ##User\\n
24 1 goal\\n
25 \\n
26 A, B, C : Prop\\n
27 A_to_B_to_C_is_true : A -> B -> C\\n
28 =====\\n
29 (A -> B) -> A -> C\\n

```

```

29 ##Assistant\n
30 move=> A_to_B_is_true.\n
31 ##User\n"

```

次に答えを生成したが不正解した例 12 を記載する．この例での正解は 2 である．

データの具体例 12 大規模言語モデルが答えを生成したが不正解した際の出力例 1

```

1  "'##System\\n
2  Require Import ssreflect.\\n
3  Theorem syllogism : forall A B C :Prop,(A -> B) /\ \ \ \ (B -> C) -> (A -> C).\\n
4  Proof.\\n
5  ##User\\n
6  1 goal\\n
7  \\n
8  =====\\n
9  forall A B C : Prop, (A -> B) /\ \ \ \ (B -> C) -> A -> C\\n
10 ##Assistant\\n
11 ``Select the best tactic as the next instruction to be given to Coq from the following
   .\\n
12 The choices are\\n
13 1. case.\\n
14 2. move=> A B C.\\n
15 3. apply:conj.\\n
16 Answer choice number is:'1'.\\n
17 ##User\\n
18 1 goal\\n
19 \\n
20 =====\\n
21 forall A B C : Prop, (A -> B) /\ \ (B -> C) -> A -> C\\n
22 ##Assistant\nmove=> A B C.\n
23 ##User\\n
24 1 goal\\n
25 \\n
26 A, B, C : Prop\\n
27 =====\\n
28 (A -> B) /\ \ (B -> C) -> A -> C\\n
29 ##Assistant\\n
30 case.\n###"

```

次に答えを生成しなかった例 13 を記載する．なお，この例での正解は 3 である．

データの具体例 13 大規模言語モデルが答えを生成しなかった際の出力例 2

```

1  "\"##System\\\\\\n
2  Require Import ssreflect.\\\\\\n
3  Set Implicit Arguments.\\\\\\n
4  Unset Strict Implicit.\\\\\\n
5  Unset Printing Implicit Defensive.\\\\\\n
6  Section logic.\\\\\\n
7  Variable A B C:Prop.\\\\\\n
8  Lemma AndOrDistL :(A/\\\\\\\\\\\\\\\\\\\\C)\\\\\\\\\\\\\\\\\\\\/(B/\\\\\\\\\\\\\\\\\\\\C)<->(A\\\\\\\\\\\\\\\\\\\\/B)/\\\\\\\\\\\\\\\\\\\\C.\\\\\\n
9  Proof.\\\\\\n
10 ##Assistant\\\\\\n
11 rewrite /iff.\\\\\\n
12 ##Assistant\\\\\\n
13 apply:conj.\\\\\\n
14 ##Assistant\\\\\\n
15 -\\\\\\n
16 ##Assistant\\\\\\n
17 case.\\\\\\n
18 ##Assistant\\\\\\n
19 +\\\\\\n
20 ##Assistant\\\\\\n
21 case=> AisTrue CisTrue.\\\\\\n
22 ##Assistant\\\\\\n

```

```

23 | by apply: conj; [apply:or_introl|].\\n\\n
24 | ##Assistant\\n\\n
25 | +\\n\\n
26 | ##Assistant\\n\\n
27 | case=> BisTrue CisTrue.\\n\\n
28 | ##User\\n\\n
29 | 1 goal\\n\\n
30 | \\n\\n
31 | A, B, C : Prop\\n\\n
32 | BisTrue : B\\n\\n
33 | CisTrue : C\\n\\n
34 | =====\\n\\n
35 | (A \\n\\n\\n\\n\\n\\n B) /\\n\\n\\n\\n\\n C\\n\\n\\n
36 | ##Assistant\\n\\n
37 | ``Select the best tactic as the next instruction to be given to Coq from the following
   | .\\n\\n
38 | The choices are\\n\\n
39 | 1. apply CisTrue.\\n\\n
40 | 2. apply BisTrue.\\n\\n
41 | 3. by apply: conj; [apply:or_intror|].\\n\\n
42 | Answer choice number is:' +n##Assistant\\n
43 | ##User\\nNo more goals.\\n
44 | ##Assistant\\n
45 | Qed.\\n
46 | End logic.\\n
47 | End A B C.\\n
48 | End Prop.\\n
49 | End Set.\\n
50 | End Class.\\n
51 | End Calculation.\\n
52 | End Assistant.\\n
53 | End User.\\n
54 | End ModusPonens.\\n
55 | End ModusHypens.\\n
56 | End ModusAndOrDistL.\\n
57 | End ModusDistL.\\n
58 | End ModusOrDistL.\\n
59 | End Mod"

```

6.5 考察

LoRA で学習を行ったモデルは、元の base model とスコアがほとんど変わらず、数字を生成しなかった個数が base model よりも悪いという結果となった。一方、Full Fine-Tuning で調整したモデルの方は約 2 倍の個数正解しているという結果となった。また、Full Fine-Tuning で調整したモデルはランダムに回答を選んだ確率の割合よりも良いスコアを出した。

正解の生成に成功したものはデータの具体例 11 のように、正解の選択肢の数字を出力するだけではなく、続けて正解の tactic を出力しているものがある。これは期待される出力そのままである。

失敗例は大きく二つ分けられる。選択肢を出力したが間違えたものと選択肢を出力しなかったものである。さらに細かく分けると、選択肢を出力したが間違えたものは tactic にならない選択肢を選んだものと tactic のタイプミスを選んだもの、tactic ではあるが今回は最適でないものを選んだものに分けられる。答えを生成したが不正解した例 12 は

tactic ではあるが今回は最適でないものを選んだものである。また、選択肢を出力しなかったものは、続く文に関係のない文や Coq のゴールを返したものと選択肢の番号を出力しなかったが、tactic または tactic に近いものを出力しているものに分けることができる。答えを生成しなかった例 13 では、正解ではないが tactic の一つを出力していた。

以上の結果から、今回のタスクである Coq のゴールに対して、最適な tactic を大規模言語モデルが出力をするには LoRA で学習では変化せず、Full Fine-Tuning で調整する必要があるとわかった。また、第 5 章で作成した、対話形式のデータセットは形式言語を使用した定理証明を目的とした大規模言語モデルへの追加学習に効果的であることがわかった。

7 実験 III

この実験では、学習データに含まれない定理の証明ができるかと学習の順番を変化させると結果が変わるのかについて検証した。

7.1 データセット

学習用のデータセットは第 6.1 節で用いたデータと同じデータセットを使用した。

学習用のデータセットに含まれない新しい定理や証明でテストデータセットを作成した。テストデータの形式は選択問題形式と同じものにした。テストデータに使用した定理は以下である。

- $\text{False} \rightarrow P$
- $\neg P \rightarrow P \leftrightarrow P \vee \neg P$
- $A \wedge B \rightarrow B \wedge A$
- $(\forall x, y, Pxy) \rightarrow (\forall y, x, Pxy)$
- $(P \rightarrow Q) \rightarrow (Q \rightarrow R) \rightarrow P \rightarrow R$
- $\neg \text{False}$
- $P \rightarrow \neg \neg P$
- $(P \rightarrow Q) \rightarrow \neg Q \rightarrow \neg P$
- $P \wedge (Q \wedge R) \rightarrow (P \wedge Q) \wedge R$
- $P \wedge (Q \vee R) \rightarrow (P \wedge Q) \vee (P \wedge R)$
- $P \rightarrow \neg P \rightarrow Q$
- $\neg(\neg P \wedge \neg Q) \rightarrow P \vee Q \leftrightarrow P \vee \neg P$

データセットのファイルサイズは解答を含めて 71 KB である。データのサンプル数は、88 である。入力単語数の平均は、677 である。なお、選択肢の数はすべて 3 つである。

7.2 実験設定

実験の仕方は使用するデータを学習データでそのままとテストデータで検証した。手順は第 6 章実験 II と同様に行った。

学習の順番を対話形式のデータから学習した後選択問題形式で学習したモデル、選択問題形式のデータから学習した後対話形式で学習したモデル、対話形式のデータと選択問題形式を混ぜてシャッフルした後一度に学習したモデルで変化があるか比べた。各学習時の設定、学習に要した時間を以下の表 5 にまとめた。

表 5 学習の設定

model 名	model F1	model F2	model F3	model F4
1 回目の学習のデータセット	対話形式	対話形式	選択問題形式	対話形式と選択問題形式
2 回目の学習のデータセット	なし	選択問題形式	対話形式	なし
base model	meta-llama/ Llama-2-7b-hf	meta-llama/ Llama-2-7b-hf	meta-llama/ Llama-2-7b-hf	meta-llama/ Llama-2-7b-hf
epoch	10	10	10	10
batch	2	2	2	2
学習率	2e-5	2e-5	2e-5	2e-5
optim	paged_ adamw_32bit	paged_ adamw_32bit	paged_ adamw_32bit	paged_ adamw_32bit
lr_ scheduler_type	"constant"	"constant"	"constant"	"constant"
max_seq_length	None	None	None	None
1 回目の学習に要した時間	44 min	52 min	1h 42 min	1h 58 min
2 回目の学習に要した時間	なし	1h 42 min	17 min	なし
学習に要した合計時間	44 min	2h 34 min	1 h 59 min	1 h 58 min

7.3 検証方法

第 6 章実験 II と同様に、大規模言語モデルに選択問題を解かせ、その正解率を評価点とした。大規模言語モデルが選択問題を完答できるか評価する。大規模言語モデルが正解の選択肢を選んだら 1、それ以外の選択肢を出力した場合や選択肢を出力しない場合は 0 とする。この評価点の平均を取り、その値を評価指標の数値とする。最大値は 1 である。

7.4 実験結果

まず、学習用データセットでの結果をまとめる．ランダムに選択肢を選んだ場合の確率の割合と元モデルのスコアを表 6 にまとめた．

表 6 学習用データセットでの基準のスコア

	ランダムに選択肢を選んだ 場合の確率の割合	meta-llama/ Llama-2-7b-hf
推論に要した 平均時間		4 sec
score	0.29	0.13
問題数	188	188
正解数	55	25
不正解数		69
数字を 生成しなかった数		94

各学習時の設定ごとの推論結果の各スコアを表 7 にまとめた．学習に用いた元のモデルは meta-llama/Llama-2-7b-hf である．スコアは四捨五入して小数点第二位までとした．

次に、テスト用データセットでの結果をまとめる．ランダムに選択肢を選んだ場合の確率の割合と元モデルのスコアを表 8 にまとめた．

各学習時の設定ごとの推論結果の各スコアを表 9 にまとめた．学習に用いた元のモデルは meta-llama/Llama-2-7b-hf である．スコアは四捨五入して小数点第二位までとした．

7.5 考察

学習用データセットで検証した場合対話形式のデータと選択問題形式のデータで Fine-Tuning で調整したモデルはいずれも 8 割以上正解した．新しく作成したテストデータセットで検証した場合、対話形式のデータと選択問題形式のデータで Fine-Tuning で調整したモデルの方はいずれも元のモデルと比べ正解数が増えた．対話形式のデータセットのみでは、正解数が他の学習したモデルに比べて少ない．

学習用データでの結果は、対話形式のデータと選択問題形式のデータの学習の順番によって正解数の差があった．テストデータでの結果は、対話形式のデータと選択問題形式のデータの学習の順番によって正解数には大きな差はなかった．そして、学習の順番を変

表 7 学習用データセットでの結果

modele 名	model F1	model F2	model F3	model F4
1 回目の学習の データセット	対話形式	対話形式	選択問題形式	対話形式と 選択問題形式
2 回目の学習の データセット	なし	選択問題形式	対話形式	なし
推論に要した 平均時間	4 sec	4 sec	4 sec	4 sec
score	0.30	0.84	0.71	0.91
問題数	188	188	188	188
正解数	57	157	133	172
不正解数	111	31	55	16
数字を 生成しなかった数	20	0	0	0

表 8 テスト用データセットでの基準のスコア

	ランダムに選択肢を選んだ 場合の確率の割合	meta-llama/ Llama-2-7b-hf
推論に要した 平均時間		4 sec
score	0.33	0.08
問題数	88	88
正解数	29	7
不正解数		43
数字を 生成しなかった数		38

表 9 テスト用データセットでの結果

modele 名	model F1	model F2	model F3	model F4
1 回目の学習の データセット	対話形式	対話形式	選択問題形式	対話形式と 選択問題形式
2 回目の学習の データセット	なし	選択問題形式	対話形式	なし
推論に要した 平均時間	4 sec	4 sec	4 sec	4 sec
score	0.34	0.51	0.49	0.48
問題数	88	88	88	88
正解数	30	45	43	42
不正解数	53	43	45	46
数字を 生成しなかった数	5	0	0	0

えると学習にかかる時間が変わった．選択問題形式から学習したほうが学習にかかる時間が大幅に減った．

以上の結果から，第 5 章で作成した，対話形式のデータセットとその選択問題形式のデータセットは学習用データセットに含まれないデータに対しても元モデルよりも正解を出力することができた．そのため，形式言語を使用した定理証明を目的とした大規模言語モデルへの追加学習に効果的であることがわかった．また，対話形式のデータセットのみでは，選択問題に対応できない．データを学習させる順番は学習にかかる時間が大きく変わり，性能にも影響がある可能性がある．選択問題形式の方が対話形式のデータより情報量が多いため，選択問題形式から学習したほうが学習にかかる時間が大幅に減ると考えられる．

8 まとめ

本研究では，Coq のゴールに対して，最適な tactic を大規模言語モデルが出力できるか検証した．具体的には，Coq の状態であるゴール情報から言語モデルに Coq の次の tactic を提案させた．

8.1 考察

第5章の実験Iから、ただ大量のデータセットを用意し、一般的に用いられるプロンプトに当てはめて学習させる方法では効果が期待できないことがわかった。特に、大規模言語モデルには Coq の tactic と Coq のゴールを判別できていないという結果が得られた。

第6章の実験IIから、今回のタスクである Coq のゴールに対して、最適な tactic を大規模言語モデルが出力をするには LoRA で学習では変化せず、Full Fine-Tuning で調整する必要がある。また、第5章で作成した、対話形式のデータセットは形式言語を使用した定理証明を目的とした大規模言語モデルへの追加学習に効果的である。

第7章の実験IIIから、第5章で作成した、対話形式のデータセットとその選択問題形式のデータセットは学習用データセットに含まれないデータに対しても元モデルよりも正解を出力することができた。そのため、形式言語を使用した定理証明を目的とした大規模言語モデルへの追加学習に効果的であることがわかった。また、対話形式のデータセットのみでは、選択問題に対応できない。データを学習させる順番は学習にかかる時間が大きく変わり、性能にも影響がある可能性がある。選択問題形式の方が対話形式のデータより情報量が多いため、選択問題形式から学習したほうが学習にかかる時間が大幅に減ると考えられる。

以上から、学習方法とデータ形式についての知見が得られた。

まず、学習方法についての知見について整理する。自然言語の場合、LoRA でもある程度精度が上がるということが知られている [2]。今回の Coq に関するデータでは、LoRA で調整したモデルは元のモデルとほとんど変わらず、Full Fine-Tuning で調整したモデルは精度が上がった。これは、「LoRA は、新しい能力を与えることが難しく、形式を整えるのみ」と言われることを補強する結果である。大規模言語モデルに Coq に関する情報が十分含まれておらず、新しい知識を加えるには、Full Fine-Tuning が必要だと思われる。

次にデータ形式についての知見について整理する。大規模言語モデルへの学習データは、主に自然言語での文章 [14] やプログラム [3] などがある。定理証明支援系の学習には、プログラムだけではなく、それに対応する返答の情報（ゴール）が必要である。今回は Coq と大規模言語モデルを人間とエージェントの対話と見立てて、対話形式でのデータを作成し、学習させる手法が有効であることを確認した。これにより、追加で学習するには、少量でも質の良いデータセットを使用することが良いことが Coq に関するデータでも同様であることが確認できた。

本手法では、形式言語を使用した定理証明を目的とした大規模言語モデルへの追加学習に効果的であることが示せた。

8.2 課題

今後は、現在の実験の選択問題形式のみではなく、一から tactic を生成させる点で精度を上げることを目指す。

精度を上げる実験方法として以下の候補が考えられる。

- (1)今回作成したデータセットの数が少ないため、様々な種類、分野のデータの数を増やす。
- (2)学習時の token の長さが制限されていることが結果に影響を及ぼしている可能性があるため、token 長を長くして比べる。
- (3)何回学習を繰り返したら良いのか (epoch を変化させる)。
- (4)各パラメータ (学習率等) を変化させて最適な学習を行う。

今回はハイパーパラメータの細かい調整を行っていないため、各モデル、データセットに合わせたハイパーパラメータ調整が必要である。

- (5)Full Fine-Tuning を LoRA に変えて性能を上げる方法を探す。
- (6)学習元の model を変えて比較する。

現在、事前学習済みのさまざまなモデルが公開されているため、形式言語を使用した定理証明を目的とした大規模言語モデルへの追加学習に最適なモデルの比較を行う。

- (7)RAG を試す。
- (8)強化学習をする。

8.3 将来研究

適切な大規模言語モデルが最適な tactic を選択する精度をさらに上げることができれば、強化学習によってさらに学習を進めることができる。

また、今回の研究のように次の tactic を予測するだけでなく Coq とのやりとりを通して大規模言語モデルが証明を完成させることを期待できる具体的には、Coq の”入力スクリプト”とそれを Coq に入力したあとの Coq のゴール情報から大規模言語モデルが tactic を提案し、Coq が検証する。その検証の結果適切でない tactic ならば別の tactic を提案させ、検証の結果 Coq のゴールが変化すれば次の tactic を提案させる。これを Coq のゴールが”No more goals.”、つまり証明が完了するまで繰り返す。これにより、数学の定理証明が Coq と大規模言語モデル間のやりとりのみで成功すれば自動定理証明が可能となることが期待される。

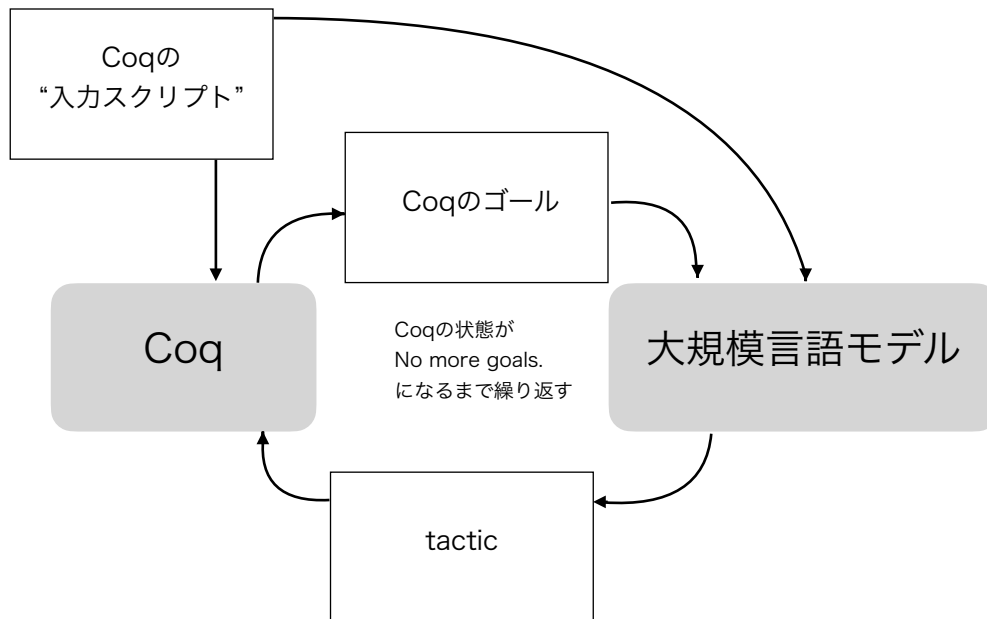


図 8 将来の手法の構造図

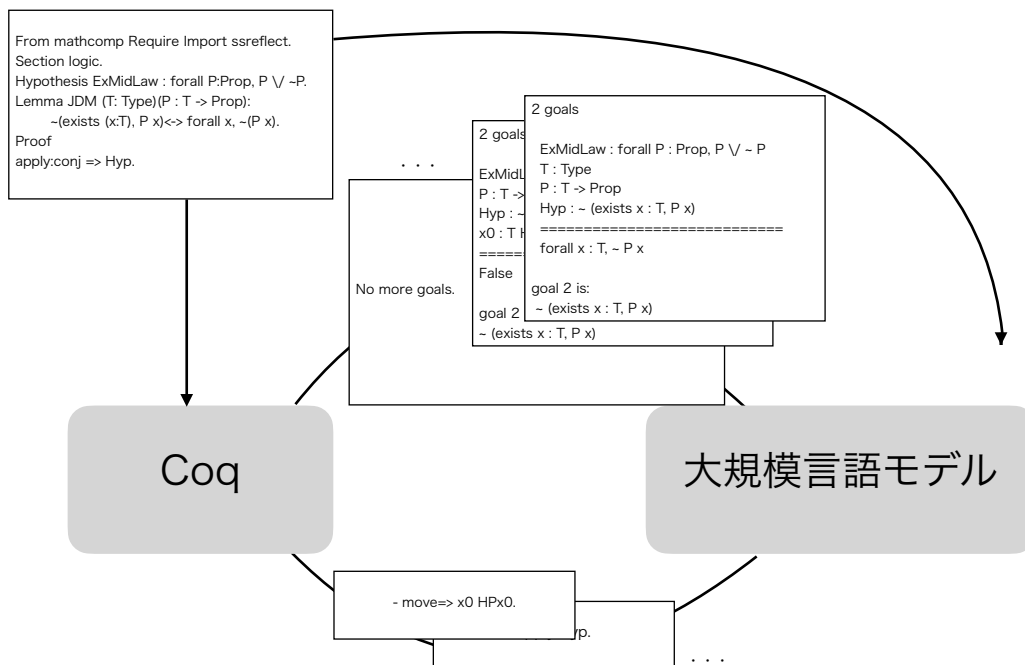


図 9 将来の手法の処理図

謝辞

指導教官の宮部賢志准教授には、研究を進めるにあたり様々な文献や論文、研究資材などの研究環境を整えて頂くとともに普段の指導や研究集会への参加を通して数学や計算論、研究活動について見識を深めることができました。六年間、貴重な指導と助言を頂き深く感謝いたします。

また、明治大学理工学研究科数学専攻の先生、研究集会では他大学の先生、学生の皆様から貴重な意見をいただきました。ありがとうございました。また、研究を通じて議論にお付き合い頂いた宮部研究室の皆様に感謝いたします。

最後になりましたが、ここに至るまでの間、私を見守り激励の言葉とともに支え続けてくれた家族と祖父母、大叔母、曾祖母に心より感謝します。ありがとうございました。

参考文献

- [1] Hugging Face – The AI community building the future. <https://huggingface.co/>.
- [2] Fine-Tuning LLMs:LoRA or Full-Parameter? An in-depth Analysis with Llama 2. <https://www.anyscale.com/blog/fine-tuning-llms-lora-or-full-parameter-an-in-depth-analysis-with-llama-2>, 2023.
- [3] Baptiste Rozière and Jonas Gehring and Fabian Gloeckle and Sten Sootla and Itai Gat and Xiaoqing Ellen Tan and Yossi Adi and Jingyu Liu and Romain Sauvestre and Tal Remez and Jérémy Rapin and Artyom Kozhevnikov and Ivan Evtimov and Joanna Bitton and Manish Bhatt and Cristian Canton Ferrer and Aaron Grattafiori and Wenhan Xiong and Alexandre Défossez and Jade Copet and Faisal Azhar and Hugo Touvron and Louis Martin and Nicolas Usunier and Thomas Scialom and Gabriel Synnaeve. Code Llama: Open Foundation Models for Code. *arXiv preprint arXiv:2308.12950*, 2023. Available: <https://arxiv.org/abs/2308.12950>.
- [4] Lasse Blaauwbroek, Josef Urban, and Herman Geuvers. Tactic Learning and Proving for the Coq Proof Assistant. In Elvira Albert and Laura Kovacs, editors, *LPAR23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Vol. 73 of *EPiC Series in Computing*, pp.

- 138–150. EasyChair, 2020.
- [5] Coq. theories library of Coq. <https://github.com/coq/coq/tree/master/theories>.
 - [6] 萩原学 and Reynald Affeldt. Coq/SSReflect/MathComp による定理証明. 森山出版, 2018.
 - [7] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685*, 2021. Available: <https://arxiv.org/abs/2106.09685>.
 - [8] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s Verify Step by Step. 2023.
 - [9] Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. Wizard-Math: Empowering Mathematical Reasoning for Large Language Models via Reinforced Evol-Instruct. *arXiv preprint arXiv:2308.09583*, 2023. Available: <https://arxiv.org/abs/2308.09583>.
 - [10] OpenAI. ChatGPT API Transition Guide. <https://help.openai.com/en/articles/7042661-chatgpt-api-transition-guide>.
 - [11] Markus Pantsar. Theorem Proving in Artificial Neural Networks: New Frontiers in Mathematical Ai. *European Journal for Philosophy of Science*, Vol. 14, No. 1, pp. 1–22, 2024.
 - [12] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpaca: A Strong, Replicable Instruction-Following Model. <https://crfm.stanford.edu/2023/03/13/alpaca.html>, 2023.
 - [13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc., 2017.
 - [14] 山田育矢, 山田康輔鈴木正敏, 李凌寒. 大規模言語モデル入門. 技術評論社, 東京, 2023.

- [15] Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. LeanDojo: Theorem Proving with Retrieval-Augmented Language Models. In *Neural Information Processing Systems (NeurIPS)*, 2023.

A 付録

以下では、本研究にて実行したソースコードを添付する。すべてのソースコードはGitHub (https://github.com/SayakaSuzuki47/LLM_Coq.git) にまとめた。

各スクリプトで使用した自作モジュールはソースコードの後半に記載する。

本研究で実験に用いたパラメーター一覧は、Params ファイルにまとめた。各学習方法とデータセット、モデル名で分けた。データセットは”theories” が実験 I のデータセット, ”interactive” が対話形式のデータセット, ”choise” が選択問題形式のデータセット, ”interactiveandchoise” が対話形式のデータセットと選択問題形式のデータセットを同時に学習させるときを表している。

A.1 追加学習, 推論に関するソースコード

追加学習や文字の生成をするソースコードを整理する。GitHub には, main フォルダにまとめた。

Python 1 Full Fine-Tuning で学習する

```
1 # ### パラメータ
2 #model_name
3 model_name="meta-llama/Llama-2-7b-hf"
4
5 #flow 学習させる datasetの順番を管理
6 #simpleは対話形式,"select"は選択問題形式を表している.
7 flow="simple"
8 #flow="simple_select"
9 #flow="select_simple"
10 flow="simpleandselect"
11
12 #モデルのloadの仕方
13 load_in = "fp32" # "fp32", "fp16", "int8" のいずれか
14 set_num=32
15
16 # 変数名等
17 flow_lst=flow.split("_")
18
19 #各datasetのファイルの場所を書く
20 def dataset_name(str_name):
21     if str_name == "simple" :
22         data_file_name="./data/interactive_format.json"
23     elif str_name == "select" :
24         data_file_name="./data/select_learn_data_plus_Assis.json"
25     else:
```

```

26         print("have not data")
27         data_file_name=""
28         return data_file_name
29
30 def select_dataset(i,lst):
31     i_name=lst[i]
32     if not( "and" in i_name ):
33         return(dataset_name(i_name))
34     else:
35         names=i_name.split("and")
36         name_lst=[ dataset_name(name) for name in names]
37         return name_lst
38
39 #実験番号
40 exp="exp_3/"
41 #プロンプト
42 prompt=''
43
44 #modelの保存等に関する情報
45 SFT_model_dir="./output/Fine_Tuning/"+exp+prompt+model_name.replace('/', '_')+"/"+flow+"
46 /"
47 SFT_model_name=flow+"/"+model_name
48 SFT_model_name=SFT_model_name.replace('/', '_')
49 print(SFT_model_dir)
50 print(SFT_model_name)
51
52 # ### work
53 import os
54 os.chdir('./work')
55 # ## Install
56 #!pip install wandb --upgrade
57 from transformers.trainer_utils import set_seed
58 # 乱数シードを42に固定
59 set_seed(42)
60 import time
61 from pprint import pprint
62 from datasets import load_dataset
63
64 # ## Wandb
65 #学習状況を可視化
66 import wandb
67 import os
68 wandb.login()
69 run=wandb.init(project=('Exp_3-Full-FF_'+prompt.replace('/', '_')+SFT_model_name))
70
71 #loggingを設定
72 from transformers import logging
73 logging.set_verbosity(logging.CRITICAL)
74
75 # ### load model
76 import torch
77 from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig,
78     AutoTokenizer,AutoModelForSequenceClassification,LlamaTokenizer,AutoModelForCausalLM,
79     AutoModelForSeq2SeqLM
80
81 device = "cuda" if torch.cuda.is_available() else "cpu"
82 print(device)
83
84 # モデルのキャッシュを保存するフォルダ
85 cache_dir = "./model_cache"
86 model_kwargs = {"trust_remote_code": True, "device_map": "auto", "low_cpu_mem_usage":
87     True, "cache_dir": cache_dir}
88
89 if load_in == "fp16":
90     model_kwargs["variant"] = "fp16"
91     model_kwargs["torch_dtype"] = torch.float16
92 elif load_in == "int8":
93     model_kwargs["variant"] = "int8"
94     model_kwargs["load_in_8bit"] = True

```

```

91
92 tokenizer = AutoTokenizer.from_pretrained(model_name, additional_special_tokens=['▮▮▮
93 ])
94 model = AutoModelForCausalLM.from_pretrained(model_name, **model_kwargs)
95
96 tokenizer.pad_token = tokenizer.eos_token
97 # Fix weird overflow issue with fp16 training
98 tokenizer.padding_side = "right"
99 import peft
100 from importlib import import_module
101 import psutil
102 from pprint import pprint
103 from datasets import load_dataset
104 from transformers import TrainingArguments, DataCollatorForLanguageModeling, Trainer
105 from trl import SFTTrainer, DataCollatorForCompletionOnlyLM
106
107 # # 学習1回目
108 # 学習2回目はメモリの関係で一からソースのやり直しをした。
109 # 注意: base modelではなく学習1回目で学習したモデルをダウンロードする。
110 # dataset
111 # 学習1回目はi=0, 学習2回目はi=1,
112 i=0
113 data_files=select_dataset(i,flow_lst)
114 print(data_files)
115 dataset = load_dataset('json', data_files=data_files, split="train")
116 dataset =dataset.shuffle(seed=42)
117 max_seq_length = max([len(tokenizer.tokenize(text)) for text in dataset["text"] ])
118 min_seq_length = min([len(tokenizer.tokenize(text)) for text in dataset["text"] ])
119
120 ### Leaning Parameter
121 # 学習に関するパラメータはそれぞれ別のファイルに書き込み, 呼び出す。
122 param_file_name=flow_lst[i]+'_'+model_name.replace('/', '_')
123 params=import_module(f'work.param.Fine_Tuning.{param_file_name}')
124 print(param_file_name)
125
126 ### Fine-Tuning
127 # Set training parameters
128 training_arguments = TrainingArguments(
129     output_dir=SFT_model_dir,
130     num_train_epochs=params.num_train_epochs,
131     per_device_train_batch_size=params.per_device_train_batch_size,
132     gradient_accumulation_steps=params.gradient_accumulation_steps,
133     optim=params.optim,
134     save_steps=params.save_steps,
135     logging_steps=params.logging_steps,
136     learning_rate=params.learning_rate,
137     weight_decay=params.weight_decay,
138     fp16=params.fp16,
139     bf16=params.bf16,
140     max_grad_norm=params.max_grad_norm,
141     max_steps=params.max_steps,
142     warmup_ratio=params.warmup_ratio,
143     group_by_length=params.group_by_length,
144     lr_scheduler_type=params.lr_scheduler_type,
145     report_to=params.report_to
146 )
147
148 # 学習の実行
149 trainer = SFTTrainer(
150     model=model,
151     train_dataset=dataset,
152     dataset_text_field=params.dataset_text_field,
153     max_seq_length=params.max_seq_length,
154     formatting_func=params.formatting_func,
155     data_collator=params.data_collator,
156     tokenizer=tokenizer,
157     args=training_arguments,
158     packing=params.packing,

```

```

159
160 params = 0
161 for p in model.parameters():
162     if p.requires_grad:
163         params += p.numel()
164
165 formatted_number = f"{params:,}"
166 print(formatted_number)
167 print('start')
168 time_start = time.time()
169 model.config.use_cache = False
170 trainer.train()
171 # SFT_model_dirに学習後のtokenizerを保存
172 #tokenizer.save_pretrained(SFT_model_dir)
173 # SFT_model_dirに学習後のモデルを保存
174 model.save_pretrained(SFT_model_dir)
175 time_end = time.time()
176 # convert second to hour, minute and seconds
177 end_time = (time_end - time_start)
178 hour = end_time // 3600
179 minute = (end_time % 3600) // 60
180 second = (end_time % 3600 % 60)
181 time_str=str(hour).zfill(2) + ":" + str(minute).zfill(2) + ":" + str(second).zfill(2)
182 print(time_str)
183
184 with open(SFT_model_dir+"time_data.txt", 'w') as f:
185     f.write("learn_"+str(i)+"_"+flow_lst[i]+"_"+time_str+"\n")
186
187 #メモリやGPUの開放
188 del dataset
189 del model
190 del tokenizer
191 import torch
192 torch.cuda.empty_cache()

```

Python 2 Full Fine-Tuning で学習したモデルで文字を生成する

```

1 # ### パラメータ
2 #model_name
3 model_name="meta-llama/Llama-2-7b-hf"
4
5 #flow 学習させる datasetの順番
6 flow="simple_select"
7
8 #モデルのloadの仕方
9 load_in = "fp32" # "fp32", "fp16", "int8" のいずれかを設定してください
10 set_num=32
11 prompt='GandT_cheat_sheet'
12 SFT_model_dir="./output/Fine_Tuning/"+exp+prompt+model_name.replace('/', '_')+"/"+flow+"
13 /"
14 SFT_model_name=flow+"/"+model_name
15 SFT_model_name=SFT_model_name.replace('/', '_')
16
17 # ### work
18 import os
19 os.chdir('./work')
20 # ## Install
21 #!pip install wandb --upgrade
22 from transformers.trainer_utils import set_seed
23 # 乱数シードを42に固定
24 set_seed(42)
25 import time
26 from pprint import pprint
27 from datasets import load_dataset
28
29 # ### load model
30 import torch

```

```

30 from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig,
    AutoTokenizer, AutoModelForSequenceClassification, LlamaTokenizer, AutoModelForCausalLM,
    AutoModelForSeq2SeqLM
31
32 device = "cuda" if torch.cuda.is_available() else "cpu"
33 print(device)
34
35 cache_dir = "./model_cache" # モデルのキャッシュを保存するフォルダ
36 model_kwargs = {"trust_remote_code": True, "device_map": "auto", "low_cpu_mem_usage":
    True, "cache_dir": cache_dir}
37
38
39 if load_in == "fp16":
40     model_kwargs["variant"] = "fp16"
41     model_kwargs["torch_dtype"] = torch.float16
42 elif load_in == "int8":
43     model_kwargs["variant"] = "int8"
44     model_kwargs["load_in_8bit"] = True
45
46 tokenizer = AutoTokenizer.from_pretrained(model_name, additional_special_tokens=['▯▯'])
47 #学習済みモデルのダウンロード
48 model = AutoModelForCausalLM.from_pretrained(SFT_model_dir, **model_kwargs)
49
50 tokenizer.pad_token = tokenizer.eos_token
51 # Fix weird overflow issue with fp16 training
52 tokenizer.padding_side = "right"
53
54 import peft
55 from importlib import import_module
56 import psutil
57 from pprint import pprint
58 from datasets import load_dataset
59 from transformers import TrainingArguments, DataCollatorForLanguageModeling, Trainer
60 from trl import SFTTrainer, DataCollatorForCompletionOnlyLM
61
62 import json
63
64 #text:LLMに入力したいテキスト
65 #gene_Num: 入力テキストの続きに出力したいテキストのトークン数を指定する. 入力のテキスト
    トークン数+gene_Numで計算する
66 def generate_text(text, gene_Num):
67     start_time = time.time()
68     inputs = tokenizer(
69         text,
70         return_tensors="pt"
71     )
72     max_length=len(inputs['input_ids'][0])+gene_Num
73     time_start = time.time()
74     outputs = model.generate(
75         **inputs,
76         pad_token_id=tokenizer.eos_token_id,
77         max_length=max_length,
78         repetition_penalty =1.1
79     )
80     decode_output=repr(tokenizer.decode(outputs.tolist()[0], skip_special_tokens=True))
81     text_len=len(text)
82     result_text=decode_output
83
84     end_time = int(time.time() - start_time)
85     # convert second to hour, minute and seconds
86     hour = end_time // 3600
87     minute = (end_time % 3600) // 60
88     second = (end_time % 3600 % 60)
89     time_str=str(hour).zfill(2) + ":" + str(minute).zfill(2) + ":" + str(second).zfill
    (2)
90     print(time_str)
91     return (result_text, end_time)
92

```

```

93 #生成に使用するデータの形はdict型{"text": }のリスト(テキストの個数分の大きさ)にする.
94 test_data_path=#生成に使用するデータへのパス
95 with open(test_data_path) as f:
96     test_dct_lst = json.load(f)
97
98 gen_time=0
99 i=0
100 for dct in test_dct_lst:
101     text=dct["text"]
102     result=generate_text(text,gene_Num)
103     #dictの"inference_text"に生成した文字列を格納する.
104     dct["inference_text"]=result[0]
105     gen_time=gen_time+result[1]
106     mean_time=gen_time/len(test_dct_lst)
107     # convert second to hour, minute and seconds
108     hour = mean_time // 3600
109     minute = (mean_time % 3600) // 60
110     second = (mean_time % 3600 % 60)
111     time_str=str(hour).zfill(2) + ":" + str(minute).zfill(2) + ":" + str(second).zfill(
112         2)
113     i=i+1
114     print(i)
115 print("mean_generate_time: "+time_str)
116
117 #元のファイルを上書きする
118 test_file_name=test_data_path.split('.')[1]
119 test_file_name=test_file_name.split('/')[2]
120 result_path=SFT_model_dir+test_file_name+'_'+SFT_model_name+"FF.json"
121 with open(result_path, 'w') as f:
122     json.dump(test_dct_lst, f)
123
124 #生成するのにかった時間の平均を保存する.
125 result_time_path=SFT_model_dir+test_file_name+'_'+SFT_model_name+'_'+time_data.txt"
126 with open(result_time_path, 'w') as f:
127     f.write("Test:Inference "+mean_generate_time: "+time_str+"\n")
128
129 # メモリやGPUの開放
130 del test_dct_lst
131 del model
132 del tokenizer
133 import torch
134 torch.cuda.empty_cache()

```

途中から学習をする際には、学習する部分 (trainer.train()) の実行部分を以下のようにする。

Python 3 途中から学習を再開する

```

1 #学習を再開したいところのチェックポイントのパスを書く.
2 dir_path=#モデルを保存している場所のパス
3 checkpoint = dir_path + "checkpoint-200"
4
5 print('start')
6 time_start = time.time()
7 model.config.use_cache = False
8 trainer.train(resume_from_checkpoint=checkpoint)
9 # SFT_model_dirに学習後のtokenizerを保存
10 #tokenizer.save_pretrained(SFT_model_dir)
11 # SFT_model_dirに学習後のモデルを保存
12 model.save_pretrained(SFT_model_dir)
13 time_end = time.time()
14 # convert second to hour, minute and seconds
15 end_time = (time_end - time_start)
16 hour = end_time // 3600
17 minute = (end_time % 3600) // 60
18 second = (end_time % 3600 % 60)

```

```

19 time_str=str(hour).zfill(2) + ":" + str(minute).zfill(2) + ":" + str(second).zfill(2)
20 print(time_str)
21
22 with open(SFT_model_dir+"time_data.txt", 'w') as f:
23     f.write("learn"+str(i)+" "+flow_lst[i]+" "+time_str+"\n")

```

A.2 第5章の実験Iに関するソースコード

第5章のデータセットの作成方法とそのソースコードを整理する。GitHub には、theories フォルダにまとめた。

データセットの作成方法の主な作成方法を下記に記す。Coq で”Show.” を実行すると、証明モード（Proof. と Qed. の間の部分）ではその状態でのゴールエリアが返答として出力される。その動作を利用して、ゴールエリアの取得と、証明状態にある場所はどこか特定する。

- (1)データセットに使用する Coq のソースファイルの各行を改行し、”Show.” を加える。
- (2)(1) で作成したファイルの n 行目までを実行する。
- (3)実行したものの中で、ゴールエリアの情報が得られたものを選ぶ。

具体的には、”Show.” を入れずに実行した後の返答と”Show.” を入れて実行した後の返答が一致しなければ、”Show.” で実行した際に追加でゴールエリアの情報が返ってきていることになる。

- (4)実行した n 行目のファイルを”instruction”，show. で返ってきた結果を”input” にする。
- (5)Coq スクリプトの n+1 行目を取り出し、次の tactic とする。

今回、Coq のファイルを実行するために Coqtop で実行した。

Bash 1 一行ずつ改行して Show. を加え、Coqtop で実行する

```

1  #!/bin/bash
2  #実行したいファイル名
3  file=$1
4  data_dir_path=$2
5
6  read_file="${file%.*}"
7  read_file_name=$(basename "$read_file")
8
9  #vファイルが入っているフォルダを指定する。
10 input_file_path="${data_dir_path}theories${file}"
11 #vファイルのn行目まで実行した後の返答ファイルを保存する場所を指定する。
12 output_dir_path_coq="${data_dir_path}coqtop_data${read_file}/"
13 #vファイルのn行目までとShow.実行した後の返答ファイルを保存する場所を指定する。
14 output_dir_path_coq_show="${data_dir_path}coqtop_data_show${read_file}/"
15
16 #元のvファイルが入っているフォルダと同じ構造にする。
17 #そのためにフォルダがなければ作成する。
18 mkdir -p $output_dir_path_coq
19 mkdir -p $output_dir_path_coq_show

```

```

20
21 line_num=1
22 while read line
23 do
24     file_line_name="${output_dir_path_coq}${read_file_name}_${line_num}"
25     file_line_show_name="${output_dir_path_coq_show}${read_file_name}_show_${line_num}"
26     vline_file="${file_line_name}.v"
27     vline_file_show="${file_line_show_name}.v"
28
29     #n行ごとのvファイルを作る
30     cat $input_file_path | awk "NR<=$line_num{print}" | tee $vline_file | tee
31     $vline_file_show
32     #n行ごとのvファイルにShow.を付けたファイルを作る.
33     echo "Show." >> "$vline_file_show"
34
35     output_coqtop_file="${vline_file%.*}.txt"
36     output_coqtop_show="${vline_file_show%.*}.txt"
37
38     # 出力ファイルのパスを指定してCoqtopを実行し、結果をファイルに保存する
39     cat "$vline_file" | coqtop | tee "$output_coqtop_file"
40     cat "$vline_file_show" | coqtop | tee "$output_coqtop_show"
41
42     line_num=$((++line_num))
43 done < $input_file_path

```

”instruction”，”input” に当たる部分を取り出す。

Bash 2 実行したものの中で、ゴールエリアの情報が得られたものを探す

```

1 #!/bin/bash
2 data_dir_path=#input dir path
3 #差分を保存する場所（ゴールエリアを保存する）
4 output_dir_path="${data_dir_path}input/"
5 #ファイル一覧の取得
6 #Show.を含めないデータ
7 input_dir_path="${data_dir_path}coqtop_data/"
8 #Show.を含めたデータ
9 input_dir_show="${data_dir_path}coqtop_data_show/"
10
11 files=$(find $input_dir_show -name "*.txt")
12
13 for file_show in ${files[@]}
14 do
15     #echo $file_show
16     #フォルダ作り
17     read_file="${file_show/$input_dir_show}"
18     read_file_name=$(basename "$read_file")
19     dir_name="${read_file/$read_file_name}"
20     output_dir="${output_dir_path}${dir_name}"
21     mkdir -p $output_dir
22
23     #元とshowでのtxt差分作り
24     file_origin="${input_dir_path}${dir_name}${read_file_name}/_show}"
25     output_file_path="${output_dir}${read_file_name}/_show}"
26     diff_result=$(diff "$file_origin" "$file_show")
27     if [[ -n "$diff_result" ]]; then
28         echo "$diff_result" > "$output_file_path"
29     fi
30 done

```

”output” に当たる部分を取り出す。具体的には、次の tactic (Coq スクリプトの n+1 行目) である。

Bash 3 v ファイルの n+1 行目を取得する


```

1 #!/bin/bash
2 data_dir_path=#Coqtopで実行した後の出力群を保存したフォルダ先を指定する.
3 input_dir_path="${data_dir_path}input/"
4 v_dir_path="${data_dir_path}coqtop_data/"
5 output_dir_path="${data_dir_path}response/line/"
6
7 #responseの作成(次の行)
8 #別で作成したディレクトリ一覧を取得する
9 files=$(find $input_dir_path -name "*.txt")
10
11 for file in ${files[@]}
12 do
13     read_file="${file}/${input_dir_path}"
14     read_file_name=$(basename "$read_file")
15     dir_name="${read_file}/${read_file_name}"
16     output_dir="${output_dir_path}${dir_name}"
17     mkdir -p $output_dir
18     read_file=${read_file%.*}
19     IFS="_" read -ra elements <<< "$read_file"
20     last_index=$(( ${#elements[@]} - 1 ))
21     last_element=${elements[$last_index]}
22     last_element=$((last_element + 1 ))
23     elements[$last_index]=$last_element
24     v_read_file=$(IFS="_"; echo "${elements[*]}")
25     v_file_path="${v_dir_path}${v_read_file}.v"
26     output_file_path="${output_dir_path}${read_file}.txt"
27     tail -n 1 $v_file_path > $output_file_path
28 done

```

A.3 第 6.1 節のデータセット作成に関するソースコード

具体的なデータの作成方法を整理する．なお，作成した Coq のスクリプトは証明モードに当たる部分に”Show.” をすでに加えてある状態とする．GitHub では，make_data フォルダにまとめた．

対話形式のデータを作成する手順は以下である．

- (1) Coq ファイルを coqtop の lv オプションと l オプションで実行し，保存する．

lv オプションでは，実行した tactic 群とそれに対するゴールエリアの情報が出力される．l オプションでは，ゴールエリアの情報のみ出力される．これらから差分を取ることで tactic 群とゴールの情報を得ることができる．

- (2) (1) で作成したファイルから tactic 群とゴールの情報に分解し，対話形式のデータにする．
- (3) 各 Coq ファイルごとに作成した対話形式のデータを一つのデータセットにまとめる．

選択問題形式のデータを作成する手順は以下である．

- (4) 対話形式のデータの (2) で作成したファイル群の Assistant の部分を選択問題の形式にする．
- (5) 選択問題は別で作成して入力する．

(6)作成した各 Coq ファイルに対する”Assistant” ごとの選択問題とその答えをデータ形式に加工する.

Bash 4 手順 (1) のスクリプト

```
1 #データに使用するCoqファイルの場所
2 input_dir="../../data/test_coq_source"
3 #Coqtopを実行した後の出力を入れるディレクトリをそれぞれ指定する(ディレクトリは空にする)
4 output_dir="../../data/test_coq_lv"
5 output_dir_l="../../data/test_coq_l"
6
7 #input_dirからCoqファイル一覧を作成する.
8 files=$(find $input_dir -name "*.v")
9
10 # 各ファイルに対して処理を繰り返す
11 for file in ${files[@]};
12 do
13     #実行結果をinput_dirと同じ構造にするためのフォルダ名取得
14     read_file="${file/$input_dir}"
15     read_file_name=$(basename "$read_file")
16     dir_name="${read_file/$read_file_name}"
17     # ファイル名から拡張子を取り除いて.txtを追加する
18     file_name=${file/$input_dir}
19     output_file_name="${file_name%.*}.txt"
20     #実際の出力ファイル名
21     output_file=$output_dir${output_file_name}
22     output_file_l=$output_dir_l${output_file_name}
23     # 出力ファイルのパスを指定してCoqtopを実行し、結果をファイルに保存する
24     coqtop -batch -lv $file 1>$output_file &
25     coqtop -batch -l $file 1>$output_file_l &
26 done
```

Python 4 手順 (2) のスクリプト

```
1 #From the results of executing coqtop -lv and coqtop -l
2 #Process the response format of #System & #User & #Assistant.
3
4 #import library
5 import os
6 import pprint
7
8 #input dir path
9 #Directory containing the result of executing coqtop -lv
10 input_dir_path='../data/coq_lv/'
11 #Directory containing the result of executing coqtop -l
12 input_dir_path_l='../data/coq_l/'
13
14 #output dir path
15 output_dir_path='../data/coq_lv_l/'
16
17 files=[f for f in os.listdir(input_dir_path) if os.path.isfile(os.path.join(
18     input_dir_path, f))]
19
20 #process_lv_l
21 #From the results of executing coqtop -lv and coqtop -l
22 #Process the response format of #System & #User & #Assistant.
23 def process_lv_l(lv_first,lv_lst,l_first,l_lst,text_lst,texts,flag):
24     if (len(l_lst)==0)and(lv_first==l_first):
25         if flag:
26             text_lst.append(texts+lv_first)
27             text_lst.append('##Assistant'+'\n'+ ' '.join(lv_lst))
28         else:
```

```

28         text_lst.append(texts)
29         text_lst.append(lv_first)
30         text_lst.append('##Assistant'+'\n'+ ' '.join(lv_lst))
31         return(text_lst)
32     elif "Show." in lv_first:
33         lv_first=lv_first.replace('Show.','##User')
34         text_lst.append(texts)
35         text_lst.append(lv_first)
36         texts=''
37         lv_first,lv_lst=lv_lst[0],lv_lst[1:]
38         return process_lv_l(lv_first,lv_lst,l_first,l_lst,text_lst,texts,flag)
39     elif lv_first==l_first:
40         if flag:
41             texts=texts+lv_first
42         else:
43             text_lst.append(texts)
44             texts=lv_first
45             flag=True
46             lv_first,lv_lst=lv_lst[0],lv_lst[1:]
47             l_first,l_lst=l_lst[0],l_lst[1:]
48             return process_lv_l(lv_first,lv_lst,l_first,l_lst,text_lst,texts,flag)
49     elif lv_first!=l_first:
50         if flag:
51             text_lst.append(texts)
52             texts='##Assistant'+'\n'+lv_first
53             flag=False
54         else:
55             texts=texts+lv_first
56             lv_first,lv_lst=lv_lst[0],lv_lst[1:]
57             return process_lv_l(lv_first,lv_lst,l_first,l_lst,text_lst,texts,flag)
58     else:
59         raise ValueError("An exception occurred")
60
61 for file_name in files:
62     #Def file path
63     input_file_path=input_dir_path+file_name
64     input_file_path_l=input_dir_path_l+file_name
65     output_file_path=output_dir_path+file_name
66
67     #open file
68     with open(input_file_path, 'r') as f:
69         lv_lst=f.readlines()
70     with open(input_file_path_l, 'r') as f:
71         l_lst=f.readlines()
72     text_lst = ['##System'+'\n']
73
74     #process
75     texts=''
76     flag=False
77     lv_first,lv_lst=lv_lst[0],lv_lst[1:]
78     l_first,l_lst=l_lst[0],l_lst[1:]
79     result=process_lv_l(lv_first,lv_lst,l_first,l_lst,text_lst,texts,flag)
80
81     #save file
82     with open(output_file_path, 'w') as f:
83         for i in result:
84             f.write(i)

```

Python 5 手順 (3) のスクリプト

```

1 import os
2 import pprint
3 from datasets import load_dataset,load_from_disk
4
5 def make_dataset_from_files(input_dir_path,output_file_path):
6     files=[input_dir_path+f for f in os.listdir(input_dir_path) if os.path.isfile(os.
           path.join(input_dir_path, f))and f.endswith(".txt")]

```

```

7     dataset = load_dataset('text', data_files=files, split="train",sample_by= "document"
8     )
9     dataset.to_json(output_file_path)
10    print("Save"+output_file_path)
11
12    #data dir path
13    output_dir_path='../data/data_json/'
14
15    #各スクリプトごとのデータが入ったフォルダ
16    input_dir_path='../data/test_coq_lv_1/'
17    file_name='test_interactive_format.json'
18    output_file_path=output_dir_path+file_name
19    files=[input_dir_path+f for f in os.listdir(input_dir_path) if os.path.isfile(os.path.
20    join(input_dir_path, f))and f.endswith(".txt")]
21    dataset = load_dataset('text', data_files=files, split="train",sample_by= "document")
22    dataset.to_json(output_file_path)

```

対話形式のデータの (2) で作成したファイル群の Assistant の部分を選択問題の形式にする。

Python 6 手順 (4) のスクリプト

```

1  #library
2  import os
3
4  #input dir path 対話形式のデータの(2)で作成したファイル群の格納されたフォルダ名
5  input_dir_path='../data/test_coq_lv_1/'
6  #output dir path
7  output_dir_path='../data/test_select_pre_data/'
8
9  #formatting
10 def select_prompt_file(input_file_path,output_file_path):
11     with open(input_file_path, 'r') as original, open(output_file_path, 'w') as modified
12     :
13         for line in original:
14             modified.write(line)
15             if "##Assistant" in line:
16                 modified.write("````Select the best tactic as the next instruction to be
17                 given to Coq from the following.\nThe choices are\n<TODO>\n")
18                 modified.write('Answer choice number is:```\n')
19
20 files=[f for f in os.listdir(input_dir_path) if os.path.isfile(os.path.join(
21 input_dir_path, f))]
22 for file_name in files:
23     input_file_path=input_dir_path+file_name
24     output_file_path=output_dir_path+file_name
25     select_prompt_file(input_file_path,output_file_path)

```

作成した各 Coq ファイルに対する”Assistant” ごとの選択問題をデータ形式に加工する。選択問題一問につき一つのデータにする。

Python 7 手順 (6) のスクリプト

```

1  from module_py import split_SAU
2  import os
3  from datasets import Dataset
4  from collections import deque
5  import pprint
6  import re
7
8  #選択問題のテキストファイルのみ格納されているフォルダ
9  input_dir_path='../data/test_select_pre_data/'
10 output_file_path="../data/data_json/test_select_text_data.json"

```

```

11
12 # 選択問題のテキストファイルのパス一覧
13 files=[input_dir_path+f for f in os.listdir(input_dir_path) if os.path.isfile(os.path.
    join(input_dir_path, f))and f.endswith(".txt")]
14
15 texts=[]
16 for input_file_path in files:
17     SAU_dct=split_SAU.split_for_SAU(input_file_path)
18     for Assistant_num in range(len(SAU_dct["##Assistant"])):
19         select_Assistant= SAU_dct["##Assistant"][Assistant_num]
20         if '```' in select_Assistant:
21             text_lst = [None]*(2+2*(Assistant_num-1))
22             text_lst[0:2]=["##System\n",SAU_dct["##System"][0]]
23             text_lst[2::2]=["##Assistant\n"]*Assistant_num
24             text_lst[3::2] = [re.sub(r'```([^\n]*)```', '\n', item)for item in SAU_dct["
                ##Assistant"][0:Assistant_num]]
25             text_lst.append("##User\n"+SAU_dct["##User"][Assistant_num])
26             text_lst.append("##Assistant\n"+select_Assistant)
27             texts.append({"text":"".join(text_lst)})
28 dataset=Dataset.from_list(texts)
29 dataset.to_json(output_file_path)
30 print("Save"+output_file_path)

```

選択問題形式のデータセット (Python コード 7 で加工した形) をテストデータに加工する。

Python 8 テストデータに関する手順 (6) のスクリプト

```

1 import os
2 from collections import deque
3 import pprint
4 import re
5 import json
6 import ast
7 import pprint
8
9 #input file path
10 input_dir_path='../data/data_json/'
11 input_file_name='AAA.json'
12
13 #output dir path
14 output_dir_path='../data/data_json/'
15 output_file_name='BBB.json'
16 output_file_path=output_dir_path+output_file_name
17
18 def split_text_Num(text):
19     # 数字で文字列を分割
20     pattern = r':\s*(\d+)\s*```'
21     matches = re.split(pattern, text)
22     # 分割されたリストから空の要素を削除
23     result = [match.strip() for match in matches if match.strip() != '']
24     return result
25
26 input_file_path=input_dir_path+input_file_name
27
28 select_data_lst=[]
29 with open(input_file_path) as f:
30     for line in f:
31         text_dct=json.loads(line)
32         split_lst=split_text_Num(text_dct["text"])
33         reault_dct={"text":repr(split_lst[0]+":."), "ans_num":split_lst[1], "ans_text":repr(
            (split_lst[2]), "inference_num":",", "inference_text":""}
34         select_data_lst.append(reault_dct)
35
36 with open(output_file_path, 'w') as f:
37     json.dump(select_data_lst, f)

```

A.4 第6章の実験IIに関するソースコード

選択問題を解かせ、それを採点するためのソースコードを整理した。GitHub では、scoring フォルダにまとめた。

選択した数字とそれ以降に生成されたテキストに分けて、正解と選択した数字が一致するか採点する。

Python 9 選択問題の採点

```
1 import json
2 import re
3 import pprint
4 from module_py import file_processing
5
6 #input
7 input_dir_path='../data/data_json/'
8 input_file_name='AAA.json'
9
10 def score_select_test(test_result):
11     Q_num=len(test_result)
12     max_score=Q_num
13     score=0
14     mis_gen=0
15     mis_gen_lst=[]
16     mis_lst=[]
17     for i in range(Q_num):
18         result=test_result[i]
19         text=result["inference_text"]
20         pattern = r'Answer_\uchoice_\u number_\u is:'
21         splited_text = re.split(pattern, text)
22         pattern=r'(?:\d+)?\s*goal'
23         splited_text = re.split(pattern, splited_text[1])
24         num_splited=re.split(r'(\d+)', "".join(splited_text[0]))
25         inference_ans=''
26         #選択した数字を生成したかどうか判定する
27         if len(num_splited)<2:
28             test_result[i]["inference_num"]="0"
29             mis_gen+=1
30             mis_gen_lst.append(i)
31         else:
32             test_result[i]["inference_num"]=num_splited[1]
33             inference_ans="".join(num_splited[2:])
34             if result["ans_num"]==num_splited[1]:
35                 score+=1
36             else:
37                 mis_lst.append(i)
38             #選択した数字を入力する
39             test_result[i]['inference_ans']=inference_ans
40     return(test_result,max_score,score,mis_gen,mis_gen_lst,mis_lst)
41
42 input_file_path=input_dir_path+input_file_name
43 with open(input_file_path) as f:
44     test_result = json.load(f)
45
46 result=score_select_test(test_result)
47 #選択した数字等を格納する
48 with open(input_file_path, 'w') as f:
49     json.dump(result[0], f)
50
51 print(input_file_name)
```

```

52 print("Question_number_is:" + str(result[1]))
53 print("score:" + str(result[2]))
54 print("Not_generate:" + str(result[3]))
55 print(result[4])
56 print("Miss")
57 print(result[5])

```

A.5 モジュール

使用したモジュールを整理する．GitHub では，module_py フォルダにまとめた．

Python 10 文を System,User,Assistant それぞれの部分に分ける

```

1 import re
2
3 #split for ##System and ##User and ##Assistant
4 def split_for_SAU(file_path):
5     with open(file_path) as f:
6         pattern = re.compile(r"##System\n|##User\n|##Assistant\n")
7         SAU_lst=re.split(pattern,f.read())
8         SAU_dct={"##System":SAU_lst[1],"##User":SAU_lst[2:2],"##Assistant":SAU_lst
9                 [3:2]}
10    return SAU_dct

```

A.6 その他

GitHub には others フォルダにまとめた．

選択問題形式ではなく，tactic を生成させるためには，生成させたい tactic のヒントとなるゴール直後の”Assistant” で区切る．この”Assistant” の続きとして tactic を生成させる．

Python 11 tactic を生成させるデータの作成

```

1 from module_py import split_SAU
2 import os
3 from datasets import Dataset
4 from collections import deque
5 import pprint
6 import re
7 import json
8
9 #各Coqファイルごとの対話形式のテキストファイルを入れたフォルダを指定する
10 input_dir_path='../data/coq_lv_1/'
11 output_file_path='../data/data_json/interactive_data.json'
12
13 files=[input_dir_path+f for f in os.listdir(input_dir_path) if os.path.isfile(os.path.
14         join(input_dir_path, f))and f.endswith(".txt")]
15 texts=[]
16 for input_file_path in files:
17     SAU_dct=split_SAU.split_for_SAU(input_file_path)
18     Assis_len=len(SAU_dct["##Assistant"])
19     for Assistant_num in range(Assis_len):
20         select_Assistant= SAU_dct["##Assistant"][Assistant_num]
21         if not(any( flag in select_Assistant for flag in ['+', '-', '*', 'Qed.'])):
22             text_lst = [None]*(2+2*Assistant_num)
23             text_lst[0:2]=["##System\n",SAU_dct["##System"][0]]
24             text_lst[2:2]=["##Assistant\n"]*Assistant_num

```

```

24     text_lst[3::2] = SAU_dct["##Assistant"][0:Assistant_num]
25     text_lst.append("##User\n"+SAU_dct["##User"][Assistant_num])
26     text_lst.append("##Assistant\n")
27     ans_len=Assis_len-Assistant_num-1
28     ans_con_text=[None]*(4*ans_len)
29     ans_con_text[0::4]=["##User\n"]*(ans_len)
30     ans_con_text[1::4]=SAU_dct["##User"][Assistant_num+1:]
31     ans_con_text[2::4]=["##Assistant\n"]*(ans_len)
32     ans_con_text[3::4] = SAU_dct["##Assistant"][Assistant_num+1:]
33
34     #"text"には入力するテキスト
35     #"ans_text"には続く正解のtactic
36     #"ans_con_text"は"ans_text"のtactic以降を格納する
37     texts.append({"text":"".join(text_lst),"ans_text":select_Assistant,"
        ans_con_text":"".join(ans_con_text),"inference_text":"","inference_ans":"","
        "inference_con":"","score":""})
38 with open(output_file_path, 'w') as f:
39     json.dump(texts, f)

```

tactic を生成させたテキストを分割して tactic とそれ以降のテキストを保存する。
tactic を生成させたテキストは各 dict の中の”inference_text”に格納されているとする。

Python 12 生成されたテキストを分割して保存

```

1  import json
2  import re
3  import pprint
4  import os
5  from module_py import file_processing
6  #input
7  input_dir_path='../data/data_json/'
8
9  # #input file name
10 input_file_name=#生成した後のデータが保存されたjsonファイルの場所
11
12 def split_test(test_result):
13     Q_num=len(test_result)
14     for i in range(Q_num):
15         result=test_result[i]
16         inference_text=result['inference_text']
17         flag=r"##System"
18         splited_text=re.split(flag,inference_text)
19         flag=r"##Assistant"
20         splited_text=re.split(flag,splited_text[1])
21         inference_ans=splited_text[1]
22         inference_con=flag+flag.join(splited_text[2:])
23         #新しく出力された部分の最初の##Assistantまでを生成されたtacticとする。
24         result['inference_ans']=inference_ans
25         #'inference_ans'以降の生成部分
26         result['inference_con']=inference_con
27     return(test_result,Q_num)
28
29 input_file_path=input_dir_path+file_path
30 with open(input_file_path) as f:
31     test_result = json.load(f)
32 result=split_test(test_result)
33 with open(input_file_path, 'w') as f:
34     json.dump(result[0], f)

```

データセットに直接プロンプトを適用する際に下記のソースコードを使用した。

Python 13 テキストデータにプロンプトを適用する

```
1 import os
2 import json
3 import re
4 import pprint
5 import collections
6 from datasets import load_dataset
7
8 #先頭に加えるprompt文を入力したテキストファイルを指定する.
9 input_prompt_file_name='cheat_sheet_goal_tactic.txt'
10 input_prompt_dir_path='../prompt/'
11 input_prompt_file_path=input_prompt_dir_path+input_prompt_file_name
12 with open(input_prompt_file_path) as f:
13     prompt_text=f.read()
14
15 #datasetのダウンロード
16 input_data_file_name='interactive_format.json'
17 input_data_dir_path='../data/data_json/'
18 input_data_file_path=input_data_dir_path+input_data_file_name
19 dataset = load_dataset('json', data_files=input_data_file_path, split="train")
20
21 #promptに加工する関数
22 # "text"が加工されるテキスト文のindex
23 def prompt_format(example):
24     #example["text"]がテキスト文なので適宜これを加工する関数に変える.
25     example["text"]=f"{interactive_prompt}\n"+example["text"]
26     return example
27
28 dataset = dataset.map(interactive_format)
29 output_file_path=input_data_dir_path+'cheat_sheet_interactive_format.json'
30 dataset.to_json(output_file_path)
```