

# Coq と大規模言語モデルによる定理証明

鈴木彩加

明治大学 理工学研究科 数学専攻 M2

October 31, 2023

# 定理証明支援系

**定理証明支援系** (Proof Assistant) とは、コンピュータを用いて「形式化」を行い、その検証<sup>1</sup>を行うためのソフトウェアである。[4] 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のを助けるシステムである。[5]

Coq, AutoMath, Lean, Isabelle など様々な定理証明支援系がある。今回は Coq を扱う。

---

<sup>1</sup>コンピュータによって漏れや間違いがなく証明の確認ができる。

# 目次

- ① アイデア
- ② データセット
- ③ 学習方法
  - LLM モデル
  - 追加学習方法
  - データセット
- ④ 検証方法
- ⑤ 問題点
- ⑥ 今後

# アイデア

数学の定理を機械に証明させたい。

定理証明支援系で人間が与えるヒントもコンピュータ上の言語で書かれている。

Coq からの検証結果を見れば次に与えたいヒントも機械が予測できるのではないか。



\*scratch\*

test

test\_01

すでに入力したCoqのスク립ト

```
Require Import ssreflect.
Section HilbertSAxion.
Variable A B C : Prop.
Theorem HS1 : (A -> B) -> (B -> C) ->(A-> C).
Proof.
move=> A to B is true.
move=> B to C is true.
move=> A is true.

apply: B to C is true.
apply: A to B is true.
```

by[].

Qed.

これを予測したい

```
1 goal
A, B, C : Prop
A_to_B_is_true : A -> B
A_is_true : A
_____ (1/1)
B
```

Coqからの返答

Messages

Errors

Jobs

# 大規模言語モデル

**大規模言語モデル (LLM)** とは、深層学習モデルの一種である。テキストデータを中心とした大規模なデータセットを学習している。従来の自然言語モデルと比べ、人間に近い会話が可能であり、自然言語を用いたさまざまな処理を行うことができる。

例：ChatGPT, Bing, ...

**解説** linuxでdiffを使ってファイルの差分を別のファイルに書き込みたいです

Linuxの'diff'コマンドを使用して、ファイルの差分を別のファイルに書き込む方法はいくつかあります。以下にいくつかの方法を紹介します。

1. 'diff'の結果をファイルに直接リダイレクトする方法:

```
diff ファイル1 ファイル2 > 差分ファイル
```

上記のコマンドを実行すると、'ファイル1'と'ファイル2'の差分が'差分ファイル'に書き込まれます。
1. 'diff'の結果を'patch'コマンドを使用して適用する方法:

```
diff -u ファイル1 ファイル2 > 差分ファイル
patch ファイル1 < 差分ファイル
```

上記のコマンドでは、'diff'の出力を'差分ファイル'に書き込み、それを'patch'コマンドで'ファイル1'に適用します。
1. 'diff'の結果を'patch'コマンドを使用せずに手動で適用する方法:

```
diff -u ファイル1 ファイル2 > 差分ファイル
```

上記のコマンドで差分をファイルに書き込んだ後、'差分ファイル'を開き、手動で変更を適用することができます。

これらの方法のどれを選ぶかは、具体的な要件や作業フローによって異なります。お使いの状況に合わせて、最適な方法を選んでください。

537 chars / 21 words Copy

Figure: ChatGPT



ただし、差分がなければファイルを出力しないでください



差分がない場合には、`diff` コマンドの出力を確認し、差分が存在するかどうかを判断することができます。以下に、そのような場合に差分ファイルを出力しない方法を示します。

javascript

Copy code

```
diff ファイル1 ファイル2 > 差分ファイル 2>/dev/null && [ -s 差分ファイル ] && cat 差分フ
```

上記のコマンドでは、まず`diff` コマンドの出力を`差分ファイル`にリダイレクトし、エラーメッセージを`/dev/null`に捨てます（`2>/dev/null`）。次に、`[ -s 差分ファイル ]`を使用して、差分ファイルが存在し、そのサイズがゼロでないことを確認します。最後に、`cat` コマンドを使用して差分ファイルの内容を表示します。

このコマンドは、差分が存在しない場合には何も出力せず、差分がある場合には差分ファイルの内容を表示します。

385 chars / 23 words

Copy

Figure: ChatGPT

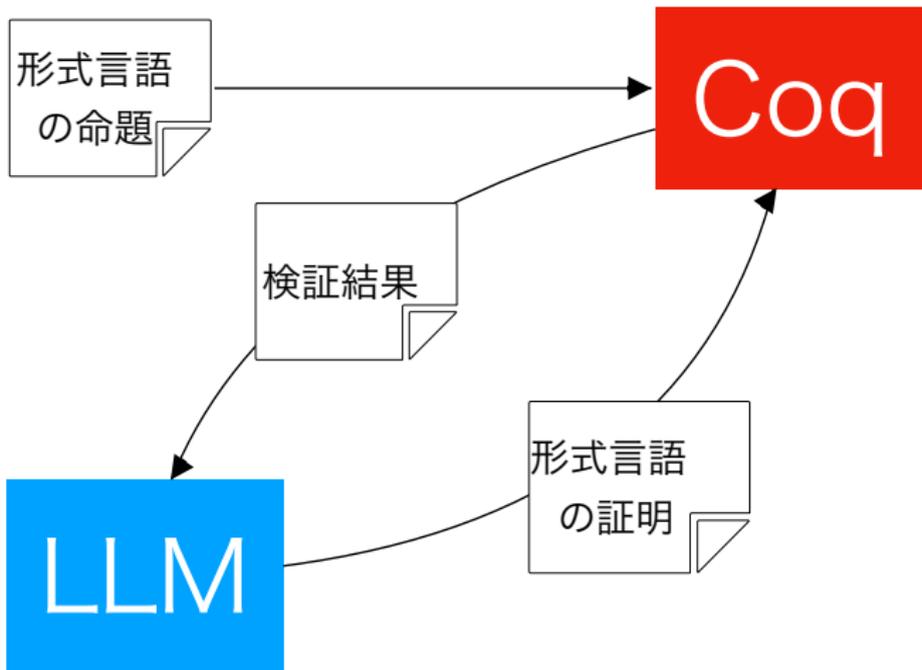


Figure: やり取りのイメージ図

# 目次

- ① アイデア
- ② データセット
- ③ 学習方法
  - LLM モデル
  - 追加学習方法
  - データセット
- ④ 検証方法
- ⑤ 問題点
- ⑥ 今後

# Alpaca 形式の Dataset [1]

```
{ "instruction" : モデルが実行するタスクを説明する,  
  "input" : タスクのオプションのコンテキストまたは入力,  
  "output" : instruction に対する応答  
}
```

例 :

```
{ "instruction" : " 次の記事を要約してください",  
  "input" : " 記事の本文",  
  "output" : " 記事の要約文"  
}
```



\*scratch\*

test

test\_01

instruction

```
Require Import ssreflect.
Section HilbertSaxion.
Variable A B C : Prop.
Theorem HS1 : (A -> B) -> (B -> C) ->(A-> C).
Proof.
move=> A to B is true.
move=> B to C is true.
move=> A is true.

apply: B to C is true.
apply: A to B is true.
```

by[].

Qed.

output

```
1 goal
A, B, C : Prop
A_to_B_is_true : A -> B
A_is_true : A
_____ (1/1)
B
```

input

Messages

Errors

Jobs

# 現在作成したデータセット

`theories` ファイルセット<sup>2</sup>から作成. (84723 個)

- Coq プログラム (拡張子 `v`) ファイルの  $n$  行目までを Coqtop で実行する.
- `show.` を最後に実行して返答があったもののみデータセットとして使用する.  
show. を最後に実行して返ってきた返答は CoqIDE 上のゴールエリアに表示されるものであり, 現在の仮定とゴールである.
- $n$  行目までを "instruction", `show.` を実行して返ってきた返答を "input",  $n + 1$  行目を "output" とする.  
つまり, 今まで出した命令を "instruction", 現在の仮定とゴールを "input" 次に出す命令を "output" として情報を与える.

---

<sup>2</sup>Coq に標準で入っている理論ライブラリ

# 目次

- ① アイデア
- ② データセット
- ③ 学習方法
  - LLM モデル
  - 追加学習方法
  - データセット
- ④ 検証方法
- ⑤ 問題点
- ⑥ 今後

# 目次

- ① アイデア
- ② データセット
- ③ 学習方法
  - LLM モデル
  - 追加学習方法
  - データセット
- ④ 検証方法
- ⑤ 問題点
- ⑥ 今後

- 多言語対応したモデル  
様々な人がモデル作成に取り組んでいるため精度が高いモデルが多くある。  
多言語対応しているため必要のない情報に情報量が割かれてしまっている。
- コードに特化したモデル  
プログラミング言語に特化したモデルであり、一番多いものは Python のスクリプトの精度を高めたものである。Python, C++, Java, PHP, Typescript(Javascript), C#, Bash などの言語をサポートしているものがある。  
例：[Code Llama](#), [WizardCoder-Python-13B-V1.0](#)
- 数学特化の言語モデル  
例：[WizardMath](#)

# 目次

- ① アイデア
- ② データセット
- ③ 学習方法
  - LLM モデル
  - 追加学習方法
  - データセット
- ④ 検証方法
- ⑤ 問題点
- ⑥ 今後

# LoRA[2]

**LoRA (Low-Rank Adaptation)** とは、学習にかかる計算コストを抑えつつ、かつ推論時の速度を Fine Tuning した場合と同等に保つことが出来る学習方法である。

LLM の事前学習から変更されていない状態の重みを  $W_0 \in R^{d \times k}$  とする。LLM を Fine Tuning して変更した層の差分を  $\Delta W$  とする。LoRA はこの  $\Delta W$  を比較的小さなサイズの行列  $A$  と  $B$  の積  $BA$  で学習する。

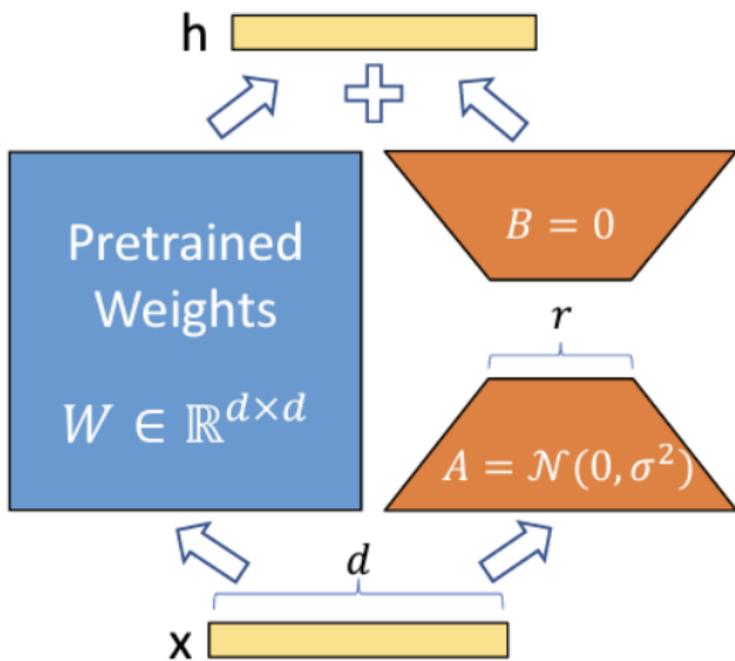


Figure: LoRA[3] の概要図

# 学習方法

- Fine Tuning
- LoRA

# 目次

- ① アイデア
- ② データセット
- ③ 学習方法
  - LLM モデル
  - 追加学習方法
  - データセット
- ④ 検証方法
- ⑤ 問題点
- ⑥ 今後

# 現在作成したデータセット

`theories` ファイルセット<sup>3</sup>から作成. (84723 個)

- Coq プログラム (拡張子 `v`) ファイルの  $n$  行目までを Coqtop で実行する.
- `show.` を最後に実行して返答があったもののみデータセットとして使用する.  
show. を最後に実行して返ってきた返答は CoqIDE 上のゴールエリアに表示されるものであり, 現在の仮定とゴールである.
- $n$  行目までを "instruction", `show.` を実行して返ってきた返答を "input",  $n + 1$  行目を "output" とする.  
つまり, 今まで出した命令を "instruction", 現在の仮定とゴールを "input" 次に出す命令を "output" として情報を与える.

---

<sup>3</sup>Coq に標準で入っている理論ライブラリ

CoqIDE

File Edit View Navigation Templates Queries Tools Compile Debug Windows Help

\*scratch\* test test\_01

```
Require Import ssreflect.
Section HilbertSAxion.
Variable A B C : Prop.
Theorem HS1 : (A -> B ) -> (B -> C ) ->(A-> C ).
Proof.
move=> A to B is true.
move=> B to C is true.
move=> A is true.

apply: A to B is true.
apply: B to C is true.

by[].
.
```

```
1 goal
A, B, C : Prop
A_to_B_is_true : A -> B
B_to_C_is_true : B -> C
A_is_true : A
----- (1/1)
C
```

Messages Errors Jobs

Cannot apply lemma A\_to\_B\_is\_true

エラー文

Ready in HilbertSAxion, proving HS1 Line: 8 Char: 18 Offset: 194 0/0



scratch FMapFacts

```

Lemma in_dec : forall m x, { In x m } + { ~ In x m }.
Proof.
  intros.
  generalize (mem_in_iff m x).
  destruct (mem x m) [left|right]; intuition.
Qed.

Lemma find_mapsto_iff : forall m x e, MapsTo x e m <=> find x m = Some e.
Proof.
  split; [apply find_1|apply find_2].
Qed.

Lemma not_find_in_iff : forall m x, ~In x m <=> find x m = None.
Proof.
  split; intros.
  - rewrite eq_option_elt. intro e. rewrite <- find_mapsto_iff.
    split; try discriminate; intro H; elim H; exists e; auto.
  - intros (e,H); rewrite find_mapsto_iff,H in H; discriminate.
Qed.

Lemma in_find_iff : forall m x, In x m <=> find x m <= None.
Proof.
  intros; rewrite <- not_find_in_iff, mem_in_iff.
  destruct mem; intuition.
Qed.

Lemma equal_iff : forall m m' cmp, Equivb cmp m m' <=> equal cmp m m' = true.
Proof.
  split; [apply equal_1|apply equal_2].
Qed.

Lemma empty_mapsto_iff : forall x e, MapsTo x e (empty elt) <=> False.
Proof.
  intuition; apply (empty_1 H).
Qed.

Lemma empty_in_iff : forall x, In x (empty elt) <=> False.

```

```

1 goal
elt, elt', elt'' : Type
m : t elt
x : key
H : ~ In x m
e : elt
----- (1/1)
MapsTo x e m -> None = Some e

```

Messages

Errors

Jobs



\*scratch\*

FMapFacts

```

Lemma map_b : forall m x (f:elt->elt'),
  mem x (map f m) = mem x m.
Proof.
intros; do 2 rewrite mem_find_b; rewrite map_o.
destruct (find x m); simpl; auto.
Qed.

Lemma map_i_b : forall m x (f:key->elt->elt'),
  mem x (map_i f m) = mem x m.
Proof.
intros;
generalize (mem_in_iff (map_i f m) x) (mem_in_iff m x) (map_i_in_iff m x f);
destruct (mem x (map_i f m)); destruct (mem x m); simpl; auto; intros.
- symmetry; rewrite <- H0; rewrite <- H1; rewrite H; auto.
- rewrite <- H; rewrite H1; rewrite H0; auto.
Qed.

Lemma map_i_o : forall m x (f:key->elt->elt'),
  (forall x y e, E.eq x y -> f x e = f y e) ->
  find x (map_i f m) = Datatypes.option_map (f x) (find x m).
Proof.
intros;
generalize (find_mapeto_iff (map_i f m) x) (find_mapeto_iff m x)
  (fun b => map_i_mapeto_iff m x b H).
destruct (find x (map_i f m)); destruct (find x m); simpl; auto; intros.
- rewrite <- H0; rewrite H2; exists e0; rewrite H1; auto.
- destruct (H0 e) as [_, H3].
  rewrite H2 in H3.
  destruct H3 as (e,_,H3); auto.
  rewrite H1 in H3; discriminate.
- rewrite <- H0; rewrite H2; exists e; rewrite H1; auto.
Qed.

Lemma map2_lbis : forall (m: t elt)(m': t elt') x
  (f:option elt->option elt'->option elt'')
  f None None = None ->
  find x (map2 f m m') = f (find x m) (find x m').
Proof.

```

```

l goal
elt, elt', elt'' : Type
m : t elt
x : key
f : key -> elt -> elt'
H : forall x y e, E.eq x y -> f x e = f y e
find x (map_i f m) = option_map (f x) [[find x m]]

```

Messages

Errors

Jobs



\*scratch\*

FMapFacts

```

men x (map f m) = men x m.
Proof.
intros; do 2 rewrite men_find_b; rewrite map_o.
destruct (find x m); simpl; auto.
Qed.

Lemma mapi_b : forall m x (f:key->elt->elt'),
men x (mapi f m) = men x m.
Proof.
intros.
generalize (men_in_iff (mapi f m) x) (men_in_iff m x) (mapi_in_iff m x f).
destruct (men x (mapi f m)); destruct (men x m); simpl; auto; intros.
- symmetry; rewrite <- H0; rewrite <- H1; rewrite H; auto.
- rewrite <- H; rewrite H1; rewrite H0; auto.
Qed.

Lemma mapi_o : forall m x (f:key->elt->elt'),
{forall x y e, E.eq x y -> f x e = f y e} ->
find x (mapi f m) = Datatypes.option_map (f x) (find x m).
Proof.
intros.
generalize (find_mapsto_iff (mapi f m) x) (find_mapsto_iff m x)
(fun b => mapi_mapsto_iff m x b H).
destruct (find x (mapi f m)); destruct (find x m); simpl; auto; intros.
- destruct <- H0; rewrite H2; exists e0; rewrite H1; auto.
- destruct (H0 e) as [_, H3].
  rewrite H2 in H3.
  destruct H3 as (a, (_, H3)); auto.
  rewrite H1 in H3; discriminate.
- rewrite <- H0; rewrite H2; exists e; rewrite H1; auto.
Qed.

Lemma map2_lbis : forall (m t elt)(m' : t elt') x
(fioption elt->option elt' ->option elt'),
f None None = None ->
find x (map2 f m m') = f (find x m) (find x m').
Proof.
intros.

```

```

l goal
elt, elt', elt'' : Type
m : t elt
x : key
f : key -> elt -> elt'
H : forall x y e, E.eq x y -> f x e = f y e
      (1/1)
{forall e : elt', Mapsto x e (mapi f m) <-> find x (mapi f m) = Some e} ->
{forall e, Mapsto x e m <-> find x m = Some e} ->
{forall b : elt', Mapsto x b (mapi f m) <-> (exists a : elt, b = f x a /\ Mapsto x a m)} ->
find x (mapi f m) = option_map (f x) [find x m]

```

Messages

Errors

Jobs

# 目次

- ① アイデア
- ② データセット
- ③ 学習方法
  - LLM モデル
  - 追加学習方法
  - データセット
- ④ 検証方法
- ⑤ 問題点
- ⑥ 今後

簡単な定理の証明について証明できるか検証する.

- 次の tactic 候補複数を LLM に提示してどれが一番良いか予測させる.
- LLM に次の tactic を予測させる.



\*scratch\*

test

test\_01

instruction

```
Require Import ssreflect.
Section HilbertSaxion.
Variable A B C : Prop.
Theorem HS1 : (A -> B) -> (B -> C) ->(A-> C).
Proof.
move=> A to B is true.
move=> B to C is true.
move=> A is true.

apply: B to C is true.
apply: A to B is true.
```

by[].

Qed.

output

```
1 goal
A, B, C : Prop
A_to_B_is_true : A -> B
A_is_true : A
_____ (1/1)
B
```

input

Messages

Errors

Jobs

次の tactic 候補複数を提示してどれが一番良いか予測させる.

- ① `apply: A_to_B_is_true.`
- ② `by[].`
- ③ `apply: A_is_true.`

LLM に次の tactic を予測させる.

tactic がエラーが出ずに証明が進めば正解とする.

可能であれば, 最適な tactic を選んでいるか評価したい. (証明終了までに必要な tactic が少ない tactic が予測出来ればスコアを高くする等)

# 目次

- ① アイデア
- ② データセット
- ③ 学習方法
  - LLM モデル
  - 追加学習方法
  - データセット
- ④ 検証方法
- ⑤ 問題点
- ⑥ 今後



\*scratch\*

```
Require Import ssreflect.
Section HilbertSaxion.
Variable A B C : Prop.
Theorem HS1 : (A -> (B -> C)) -> (A -> B) -> C.
Proof.
move=> A_to_B_to_C_is_true.
move=> A_to_B_is_true.
move=> A_is_true.

apply: A_to_B_to_C_is_true.
```

```
apply: A_is_true.
apply: A_to_B_is_true.
```

by[.]

Qed. |

予測して欲しいもの

```
2 goals
A, B, C : Prop
A_to_B_is_true : A -> B
A_is_true : A
_____ (1/2)
A
_____ (2/2)
B
```

Messages ↗

Errors ↗

Jobs ↗

### output:

3 goals

> HS1 : (A -> (B -> C) ) -> ( (A -> B )->(A-> C) ).

> A\_to\_B\_to\_C\_is\_true : (A -> (B -> C) ) -> ( (A -> B )->(A-> C) ).

> A\_to\_B\_is\_true : A -> B

> A\_is\_true : A .

# 目次

- ① アイデア
- ② データセット
- ③ 学習方法
  - LLM モデル
  - 追加学習方法
  - データセット
- ④ 検証方法
- ⑤ 問題点
- ⑥ 今後

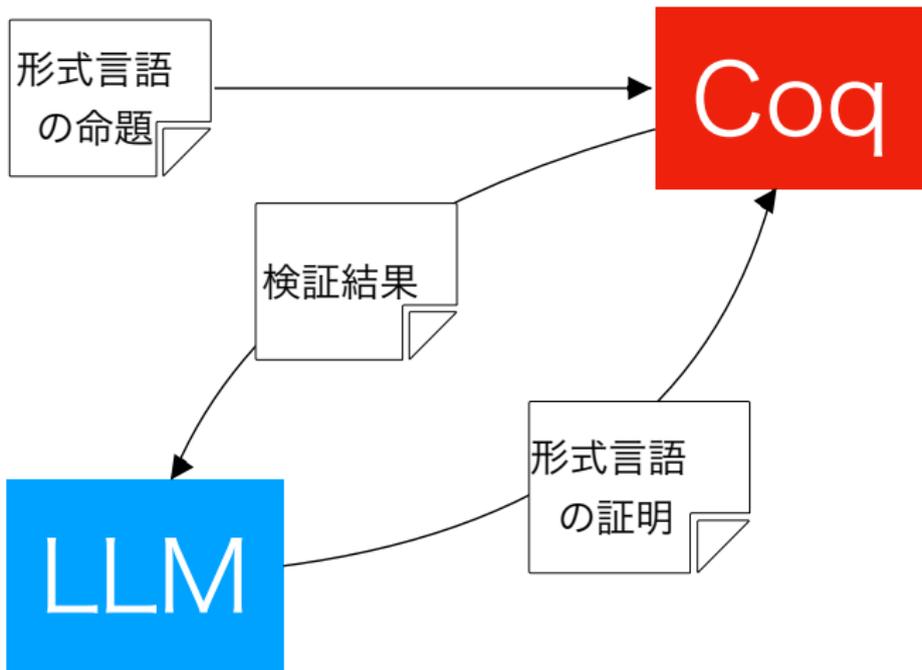


Figure: やり取りのイメージ図

Coq と LLM で交互にやり取りをさせて定理証明できるか検証したい。  
Coq からの検証結果だけでなくエラー文も含めて LLM がよりよい tactic 候補を出せるようにする。

# 参考文献 I

- [1] Alpaca dataset, 2023.  
<https://huggingface.co/datasets/tatsu-lab/alpaca>.
- [2] Llm をカスタマイズする方法として lora について調べてみました。 , 2023.  
<https://techblog.cccmk.co.jp/entry/2023/06/20/154850>.
- [3] Lora: Low-rank adaptation of large language models, 2023.  
<https://arxiv.org/abs/2106.09685>.

## 参考文献 II

- [4] 久我健一.  
証明支援系を用いたトポロジーの形式化について,  
2015.
- [5] 萩原 学 and Reynald Affeldt.  
Coq/ssreflect/mathcomp による定理証明, 2018.

終わり