LLM-jp チューニングコンペティション 2026 ヒント・手順・Python コード生成、実行による多段パイプラインを用いた数学回答能力の向上

山坂巧 [1]　橋本賢一 [2]
[1] 株式会社 DIVX

概要

本稿は LLM-jp チューニングコンペティション 2026 オープン枠への参加報告である。コンペティションの対象となる言語モデルは、数式回答精度が低い傾向にある。そこで言語モデルを基盤とするシステムとして、LLM1（ヒント生成）、LLM2（解法手順生成）、LLM3（Python/SymPy コード生成）と CodeExecutor（コード実行）から成る 3 段パイプラインを構築し評価した。各段を専用 LoRA アダプタでファインチューニングすることで役割を専門化し、計算処理を Python に委譲する設計とした。開発時の評価セットにおいて、改善前は 3 割程度に対し提案手法を適用すると約 6 割を達成し、おおよそ+25pt の改善を得た。

1　はじめに

LLM-jp チューニングコンペティション 2026 は、`llm-jp-4` 8B モデルを用い数学問題を解くシステムを構築する課題である。数学問題は中学 1 年から数学 IIIC までの範囲となっている。LLM-jp モデルを追加学習することはもちろん、推論の途中で電卓を呼び出す、複数の回答候補をサンプリングして多数決を取るなどの手法を用いてもよい。

8B 規模のモデルは自然言語推論に一定の能力を持つが、多段階計算や複雑な数式変形で誤りが生じやすい。テキストのみの直接回答では、計算ミスが精度上限を規定する。推論の途中で電卓を呼び出す手法 [1] もあるが、数学タスクでは変数を含む式変形や記号的な処理が求められるため、電卓方式では対応が困難である。He-Yueya ら [2] は、LLM が本質的に苦手とする数値計算を SymPy 等の記号計算ツールに委譲することで数学問題の正答率を改善できることを示した。

本チームではこの着想を発展させ、計算処理を Python（SymPy）へ委譲する 3 段パイプラインを提案する。8B パラメータかつコンテキスト長 8k という制約下では、問題の理解・解法設計・コード生成を単一モデルに一括で担わせることは困難である。大規模商用モデルであれば一度に実行可能な処理も、小規模モデルでは役割を分割し各段を専門化する必要がある。そこで問題理解（ヒント生成）→解法設計（手順生成）→ コード生成 → 実行という分担により、各 LoRA アダプタを専門化し、単純化された役割に集中させる設計とした。

また、Shi ら [3] は非英語タスクでも英語で中間推論を行う方が性能が高いことを示している。この知見に基づき、事前学習データの大半が英語であることを活かすため、ヒントおよび手順の生成を英語で行った。

本稿の主な貢献は次の 4 点である。

- LLM1/LLM2/LLM3 による 3 段推論アーキテクチャを設計した。
- `gpt-oss-120B` を用いて 108,710 問の合成学習データを構築した。
- 各段を LoRA ファインチューニングで専門化学習し、役割分担を実現した。
- 開発時の評価セットでベースライン比 Overall +25pt の改善を達成した。

2　提案手法

2.1　多段パイプラインの全体構成

提案手法の推論パイプラインを図 1 に示す。設計の要点は「考える」と「計算する」の分離である。LLM1 が問題を読み解きヒントを生成し、LLM2 がヒントから具体的な解法手順を設計し、LLM3 がその手順を Python/SymPy コードに変換する。CodeExecutor がコードを実行し、最終回答を抽

出する。

なお、LLM1 と LLM2 を同一モデルに統合する設計も考えられるが、本チームではあえて両者を分離した。これは LLM2 の役割を将来的に RAG（検索拡張生成）で置換し、類似問題の解法を検索・参照する方式へ拡張する可能性を残すためである。今回は LLM2 による推論で解法手順を生成する方式を採用した。

各段を軽量 LoRA で専門化することで、同一ベースモデルを共有しつつ複数の異なる役割を実現する。コード実行が失敗した場合はリトライ後にテキスト直接回答モデル（`LLM_txt`）へフォールバックし、ロバスト性を確保する。

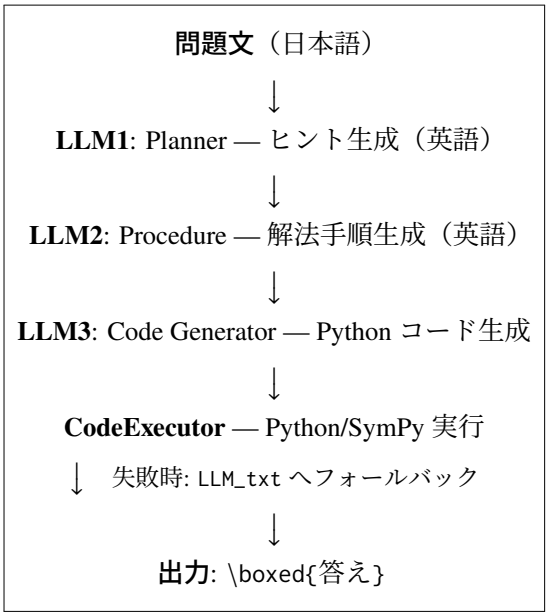

図 1　提案する 3 段推論パイプラインの全体構成。

2.2　各段の役割

各段の役割を表 1 にまとめる。LLM1 は問題文を受け取り、解法の方針となるヒントを英語で出力する。LLM2 はヒントを踏まえ、具体的な解法ステップを英語で列挙する。LLM3 は手順に従い Python/SymPy コードを生成する。コード実行が失敗した場合のフォールバックとして、`LLM_txt` が手順からテキストで直接回答を生成する。

2.3　プロンプト設計

システムプロンプトは全段共通で以下を使用する。

> 以下は、タスクを説明する指示です。要求を適切に満たす応答を書きなさい。

ユーザープロンプトは「【指示】\n{instruction}\n\n【問題】\n{body}」形式で統一した。各段の指示文の概要を以下に示す。

- **LLM1**: 数学教師として、問題を解くためのヒントを英語で生成する。
- **LLM2**: ヒントを踏まえ、解法のステップ列を英語で生成する。
- **LLM3**: コードブロックのみを出力し、変数 `answer` に最終解を代入する。`assert` 文は禁止。
- **LLM_txt**: 問題・ヒント・手順から中間式を示しつつ答えを導出する。

表 1　各段の役割。

段	モデル名	入力	出力
1	LLM1	問題文	英語ヒント
2	LLM2	問題文+ヒント	英語手順列
3	LLM3	問題文+手順	Python コード
FB	LLM_txt	問題文+手順	テキスト回答

2.4　生成パラメータ

各段の生成パラメータを表 2 に示す。LLM1・LLM2 は temperature 0.6 で多様性を持たせ、リトライ時には 0.8 に上げて異なる出力を誘導する。LLM3 はコード生成の再現性を重視し temperature 0.0 とした。

表 2　各段の生成パラメータ。括弧内はリトライ時の値。

モデル	temperature	max_tokens
LLM1	0.6 (0.8)	512
LLM2	0.6 (0.8)	2048
LLM3	0.0	2048
LLM_txt	0.0	250

2.5　コード実行エンジン

LLM3 の出力から “‘python ... “‘ブロックを正規表現で抽出し、`subprocess.run` でサブプロセスとして実行する。実行環境では `sympy`、`fractions`、`math` を自動 import し、数式処理に必要なライブラリをすぐに利用可能とした。

解の抽出では、`answer`、`final_answer`、`result` 等 13 種の変数名エイリアスを優先順で探索する。SymPy オブジェクトが得られた場合は `sympy.latex()` で LaTeX 形式に変換する。浮動小数点数は有理数近似を試行し、可能な場合は分数表現に変換する。タイムアウトは 10 秒に設定した。

2.6 リトライとフォールバック

各問題に対しパイプライン全体を最大 2 回試行する。1 回目が失敗した場合、LLM1・LLM2 の temperature を 0.8 に上げて再試行する。2 回とも失敗した場合は `LLM_txt` によるテキスト直接回答にフォールバックする。最終回答は`\boxed{...}`形式で出力する。

3 学習データ

3.1 合成データ生成

外部モデル `gpt-oss-120B` を vLLM でサーブし、次の 5 段階で合成学習データを生成した。

1. **問題生成**: トピック・学年を指定し、数学問題を日本語で生成。
2. **ヒント生成**: 問題に対する解法のヒントを英語で生成。
3. **手順生成**: ヒントを踏まえた解法ステップ列を英語で生成。
4. **解法生成**: 手順に従い Python/SymPy コードを生成。
5. **コード実行検証**: 生成コードを実行し、期待解と一致したサンプルのみ保持。

ヒント・手順・解法を英語で生成したのは、推論時の中間出力言語を学習時と統一するためである。コード実行による検証をフィルタとして用いることで、不正確なサンプルを自動的に排除した。

3.2 データ統計

トピック別・学年別の件数を表 3 に示す。6 トピック各約 10,000 問に加え、学年別（中 1〜数学 IIIC）のデータを作成した。全ファイル間で重複除去済み、合計 108,710 問である。数学 IIIC のみ 3,145 問と少ないが、これは高難度問題の検証通過率が低いためである。

3.3 訓練データへの変換

合成データを各モデル用の `messages` 形式 JSONL に変換した。各 JSONL ファイルにはシステムプロンプト、ユーザープロンプト（指示文＋問題文）、アシスタント応答（ヒント/手順/コード/テキスト回答）の 3 ターンを格納する。95:5 の比率（seed=42）で train/eval 分割した結果を表 4 に示す。LLM3 の学習データがやや少ないのは、コード実行検証で不合格となったサンプルを除外しているためである。

表 3 合成データの統計。左: トピック別、右: 学年別。

トピック	件数	カテゴリ	件数
Algebra	9,557	中 1	9,610
Calculus	8,378	中 2	9,761
Complex	8,871	中 3	9,454
Linear_Alg	9,326	数学 IA	9,881
Prob_Stat	9,920	数学 IIB	11,089
Sequence	9,718	数学 IIIC	3,145

表 4 各モデルの学習・評価データ件数。

モデル	学習	評価
LLM1 (Planner)	97,508	5,132
LLM2 (Procedure)	97,508	5,132
LLM3 (Code Gen.)	94,824	5,132
LLM_txt (Fallback)	97,508	5,132

4 学習

ベースモデル `v4-8b-decay2m-ipt_v3.1-instruct4`（LLaMA 8B）に対し、各段専用の LoRA アダプタをファインチューニングで訓練した。実装は Hugging Face TRL `SFTTrainer` と PEFT `LoraConfig` を用いた。計算環境は ABCI `rt_HG` ノード（NVIDIA A100 GPU）で、Singularity コンテナ内で実行した。

4.1 LoRA 設定

LoRA 設定を表 5 に示す。全モデル共通で、rank 16、alpha 32 とした。ターゲットモジュールは Attention 層の Q, K, V, O 射影とし、ベースモデルのパラメータは凍結したまま軽量なアダプタのみを学習する。

表 5 LoRA 設定（全モデル共通）。

パラメータ	値
rank (r)	16
alpha (α)	32
dropout	0.05
target_modules	q_proj, k_proj, v_proj, o_proj
bias	none

4.2 ファインチューニングのハイパーパラメータ

訓練ハイパーパラメータを表 6 に示す。max_seq_length は LLM1 が 512、LLM2・LLM3・LLM_txt が 1024 とした。LLM1 のヒント出力は比較的短いため、短いシーケンス長で効率的に学習できる。EarlyStopping は patience=3 とし、評価ロスが 3 エポック連続で改善しない場合に学習を打ち切った。

表 6 ファインチューニングのハイパーパラメータ。

パラメータ	値
learning_rate	2e-4
epochs	3
batch_size (per device)	4
gradient accumulation steps	4
実効バッチサイズ	16
max_seq_length	512（LLM1）/ 1024（他）
weight_decay	0.01
dtype	bfloat16
EarlyStopping	patience=3

5 推論

5.1 入出力形式

入力は 1 行 1 問の JSONL 形式で、各行は以下の構造を持つ。

```
{"id": <int>, "problem": "問題文"}
```

出力も同様の JSONL 形式で、最終回答を格納する。

```
{"id": <int>, "output": "\boxed{答え}"}
```

5.2 推論パイプラインの実行

推論は以下のコマンドで実行する。

```
singularity run submission.sif \
  --input_path input.jsonl \
  --output_path output.jsonl
```

内部処理は LLM1 でヒント生成、LLM2 で手順生成、LLM3 でコード生成、CodeExecutor で実行・解抽出の順に行う。失敗時は温度を上げてリトライ（最大 2 回）し、それでも失敗した場合は LLM_txt にフォールバックする。

5.3 提出イメージの構成

Singularity イメージにベースモデルと LLM1/LLM2/LLM3/LLM_txt の 4 つの LoRA アダプタを同梱する。ベースイメージは python:3.12-slim-bookworm とし、推論に必要なライブラリ（transformers、peft、sympy 等）を事前にインストールした。CLI は--input_path と--output_path の 2 引数を受け取る。評価環境は ABCI 3.0（H200 ×8、2TB RAM）、実行上限 500 分、ファイルサイズ上限 64GB である。

6 評価結果

6.1 評価設定

開発時の評価セットにより性能を測定した。カテゴリは中 1、中 2、中 3、数学 IA、数学 IIB、数学 IIIC の 6 種である。比較対象（ベースライン）は、LoRA アダプタなしのベースモデルによるテキスト直接回答とした。

6.2 結果

カテゴリ別の結果を表 7 に示す。

表 7 開発時の評価セットにおけるベースラインと提案手法の正答率比較。

カテゴリ	ベースライン	提案手法	改善幅
中 1	.667	.933	+.266
中 2	.667	.611	−.056
中 3	.438	.625	+.187
数学 IA	.067	.667	+.600
数学 IIB	.154	.462	+.308
数学 IIIC	.400	.600	+.200
Overall	.380	.630	+.250

提案手法は Overall で 0.380 から 0.630 へと+25.0pt の改善を達成した。

6.3 考察

数学 IA での改善（+0.600）が最も大きい。ベースラインでは正答率 0.067 とほぼ正解できていなかったが、提案手法では 0.667 まで向上した。公式の適用や計算の正確さが求められる問題が多く、コード実行による正確な計算がこれを補完した結果である。

中 1 では+0.266 と大幅に改善し、0.933 の高い正答率を達成した。比較的単純な計算問題ではコード実行の効果が顕著であることがわかる。中 3 でも+0.187 の改善が得られた。

一方、中 2 は唯一ベースラインをわずかに下回っ

た（−0.056）。ベースライン自体が 0.667 と高く、パイプラインの多段処理によるオーバーヘッドが一部の問題で悪影響を与えた可能性がある。この点は今後の分析課題である。

数学 IIB（+0.308）・数学 IIIC（+0.200）は改善したものの、絶対的な正答率は依然として 5 割前後にとどまる。微積分・複素数平面など高度な数学では、適切なコードを生成すること自体の難度が高く、コード生成品質が律速要因となっている。

総合的に、計算処理を Python に委譲することで 8B モデルは問題理解と手順設計に集中でき、Overall で+25.0pt の改善を達成した。

7 再現手順

7.1 環境構築

計算環境として ABCI（PBS、A100 GPU）と Singularity を利用する。コンテナは `pytorch:2.5.1-cuda12.1-cudnn9-runtime` をベースに、以下の主要ライブラリをインストールする。

- `transformers==4.56.1`
- `peft==0.18.0`
- `trl==0.8.6`
- `sympy`、`vllm`（データ生成用）

ベースモデルは `v4-8b-decay2m-ipt_v3.1-instruct4`（4 shard safetensors 形式）を使用する。

7.2 合成データ生成

`gpt-oss-120B` を vLLM でサーブし、5 段階（問題生成 → ヒント → 手順 → 解法 → コード実行検証）で合成データを生成する。6 トピック（Algebra, Calculus, Complex, Linear_Alg, Prob_Stat, Sequence）各約 10,000 問に加え、学年別（中 1〜数学 IIIC）のデータを作成し、重複除去後に合計 108,710 問を構築する。

7.3 ファインチューニング

LLM1、LLM2、LLM3、LLM_txt の 4 アダプタを個別に学習する。共通設定は LoRA r=16、α=32、学習率 2e−4、3 epochs、実効バッチサイズ 16 である。ABCI の `rt_HG` ノードでジョブを投入し、Singularity コンテナ内で実行する。

7.4 推論・提出

学習済みのベースモデルと 4 つの LoRA アダプタを Singularity イメージに同梱する。推論は`-input_path` と`-output_path` を指定して実行し、パイプライン全体が自動で動作する。

8 まとめ

本チームは、8B モデルの計算弱点を Python コード生成・実行への委譲で補う 3 段パイプラインを提案した。LLM1（ヒント生成）、LLM2（手順生成）、LLM3（コード生成）の段分割と LoRA 専門化学習により、ファインチューニングのみで開発時の評価セットにおいて Overall 0.380 から 0.630（+25.0pt）を達成した。

今後の課題として、数学 IIB・IIIC の高難度領域に向けたコード生成品質の向上、学習データの拡張、およびリトライ戦略の改善が挙げられる。

謝辞

本コンペティションの開催および ABCI 計算資源の提供をいただいた LLM-jp プロジェクトに深く感謝いたします。

参考文献

[1] Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., Schulman, J. Training Verifiers to Solve Math Word Problems. arXiv:2110.14168, 2021.

[2] He-Yueya, J., Poesia, G., Wang, R.E., Goodman, N.D. Solving Math Word Problems by Combining Language Models With Symbolic Solvers. arXiv:2304.09102, 2023.

[3] Shi, F., Suzgun, M., Freitag, M., Wang, X., Srivats, S., Vosoughi, S., Chung, H.W., Tay, Y., Ruder, S., Zhou, D., Das, D., Wei, J. Language Models are Multilingual Chain-of-Thought Reasoners. arXiv:2210.03057, 2022.