seq2pipe

ローカル LLM による QIIME2 パイプライン自動生成 AI エージェント

Rhizobium-gits Claude (Anthropic)

https://github.com/Rhizobium-gits/seq2pipe

2026 年 2 月 23 日

Abstract

本稿では、ローカル大規模言語モデル（LLM）を用いたマイクロバイオーム解析自動化エージェント **seq2pipe** の設計と実装について報告する。seq2pipe はユーザーが保有する生 FASTQ データを入力として受け取り、データ構造の自動認識・QIIME2 解析パイプラインの設計・実行可能シェルスクリプトの生成を行い、続いて **ツール呼び出し型コード生成エージェント**（`code_agent.py`）が実際のエクスポートファイルの内容を先に読んでからコードを生成するため、フォーマットの不一致によるエラーを根本的に排除する。コードエージェントは「NEVER GIVE UP」方針に従い、エラーが発生するたびにコードを修正して `EXIT CODE: 0` になるまで実行を繰り返す。自律モードでは 5 フェーズ・14 種類の図を全自動で生成する（クオリティ確認、多様性、多様性、分類組成、サンプル相関）。処理はすべてユーザーのローカル環境で完結し、クラウドサービスや有料 API への依存を排除する。

1 はじめに

マイクロバイオーム研究において QIIME2 [1] は 16S rRNA アンプリコンシーケンシングデータの標準的な解析プラットフォームとして広く普及している。しかし、QIIME2 の学習コストは高く、データ形式の理解・適切なパラメータ選択・Docker を介した実行環境の構築など、初心者にとっての障壁が大きい。さらに QIIME2 の出力（`.qzv` アーティファクト）の可視化には `view.qiime2.org` への依存があり、オフライン環境では解析結果の確認も困難である。

近年、大規模言語モデル（LLM）の急速な発展により、自然言語によるコード生成やドメイン特化型の解析支援が実用化されつつある [2]。一方で、従来のクラウド型 LLM サービスはコスト・プライバシー・ネットワーク依存性という問題を持つ。Ollama [4] をはじめとするローカル LLM 実行フレームワークの登場により、これらの課題をクリアしながら高度な言語処理を実現することが可能となった。

本研究では、これらの技術を統合した対話型 AI エージェント **seq2pipe** を開発し、その設計原則・アーキテクチャ・実装詳細を報告する。seq2pipe は QIIME2 解析の自動化に留まらず、Python による下流解析・統計検定・機械学習・可視化、そして日本語・英語両対応の自動レポート生成まで一貫したパイプラインを提供する点で、従来ツールを大きく超える。

2 背景と関連研究

2.1 QIIME2 における解析の複雑性

QIIME2 は、データのインポートからデノイジング（DADA2）・系統樹構築・分類学的解析・多様性解析・差次解析まで、多段階の処理を必要とする。各ステップで適切なコマンドとパラメータを選択するにはシーケンシングのライブラリ構成（ペアエンド/シングルエンド）・増幅領域（V1–V3, V3–V4, V4 など）・プライマー配列の知識が不可欠である。

2.2 LLM エージェントの台頭

ReAct フレームワーク [3] に代表されるように、LLM が思考と行動を交互に実行することで複雑なタスクを自律的に処理する「エージェント」パターンが確立されている。関数呼び出し（Function Calling）機能を持つ LLM は、外部ツールを適切なタイミングで呼び出しながら、ユーザーの意図を達成することができる。

2.3 ローカル LLM の実用化

Ollama は `llama.cpp` を基盤とし、macOS・Linux・Windows 上で量子化された LLM モデルをシングルバイナリで実行できる。OpenAI の Chat Completions API と互換性のある REST API（`/api/chat`）を提供しており、関数呼び出しを含む高度な推論が可能である。

3 システム設計

3.1 設計原則

seq2pipe は以下の設計原則に従って開発した。

1. **外部依存ゼロ**: Python 標準ライブラリ（`json`, `urllib`, `subprocess`, `pathlib`, `socket` 等）のみを使用し、`pip install` を不要とする。
2. **ローカル完結**: Ollama のローカル推論エンジンを使用し、インターネット接続を不要とする（初期セットアップを除く）。
3. **クロスプラットフォーム**: macOS・Linux・Windows のすべてで動作する。
4. **安全なコマンド実行**: コマンド実行前にユーザーへの確認を必須とする。
5. **QIIME2 ドメイン知識の内包**: システムプロンプトに QIIME2 の完全なワークフロー知識を埋め込む。
6. **多言語 UI**: 起動時に日本語 / 英語を選択し、AI 応答・自動生成レポートを統一する。
7. **堅牢なエラー処理**: 接続エラー・タイムアウト・空レスポンス・ファイルシステムエラーをすべて適切にキャッチし、ユーザーに分かりやすいメッセージを返す。

3.2 全体アーキテクチャ

図 1 に seq2pipe の 3 層アーキテクチャを示す。ユーザーは起動スクリプトを通じてエントリーポイント（`cli.py`）を起動する。`cli.py` は虹色 ASCII バナーを表示し、2 つの操作モード（指定解析モード / 自律エージェントモード）を選択させる。いずれのモードでも `pipeline_runner.py` が QIIME2 パイプラインを実行して結果をエクスポートし、続いて `code_agent.py` が vibe-local 方式のツール呼び出し型コード生成を行う。

4 実装詳細

4.1 起動シーケンスと多言語 UI

エージェントの起動時に `select_language()` が呼ばれ、ユーザーは日本語（`ja`）または英語（`en`）を選択する。選択は `LANG` グローバル変数に保存され、`ui()` 関数が全 UI テキストを選択言語で返す。自動生成されるレポートも同じ言語設定を使用する。

また `check_python_deps()` が起動時に numpy・pandas・matplotlib・seaborn の存在を `subprocess` 経由で確認し、不足する場合はインストールコマンドを案内する。

4.2 システムプロンプトへのドメイン知識埋め込み

`SYSTEM_PROMPT` 変数に QIIME2 の完全なワークフロー知識を記述した。これには以下の情報が含まれる。

- データ形式の自動判定基準（`*_R1*.fastq.gz` パターンによるペアエンド検出など）
- 全 8 ステップの QIIME2 コマンド（インポート〜差次解析）
- 増幅領域別推奨パラメータ（V1–V3, V3–V4, V4）
- Docker 実行テンプレート
- メタデータファイル形式仕様
- SILVA 138 分類階層の説明
- 一般的なエラーとトラブルシューティング
- Python 下流解析・自律探索モードの実行指針

この手法により、汎用 LLM モデルが QIIME2 専門家として振る舞うことを可能にした。

4.3 ツール定義と関数呼び出し

seq2pipe は 2 種類のツールセットを持つ。`qiime2_agent.py` が QIIME2 パイプライン生成用の 11 ツール（表 1）を提供し、`code_agent.py` が vibe-local 方式の Python コード生成用 5 ツール（表 2）を提供する。コードエージェントは、LLM がコードを書く前に必ず `read_file` でファイルの列名・形式を確認するよう指示されており、盲目的なコード生成によるフォーマットエラーを排除する。

Table 1: qiime2_agent.py のツール一覧（QIIME2 パイプライン生成）

ツール名	機能
inspect_directory	ファイル一覧・サイズ取得、FASTQ/QZA/メタデータを自動判定
read_file	テキストファイルを最大 50 行読み込む
write_file	スクリプト・マニフェスト・README を書き出す
edit_file	既存ファイルの文字列を置換する
generate_manifest	FASTQ ディレクトリから QIIME2 マニフェスト TSV を自動生成
run_command	ユーザー確認後にシェルコマンドを実行；conda 環境を自動検出・SEQ2PIPE_AUTO_YES 対応
check_system	Docker・Ollama・Python・ディスク容量を確認
set_plot_config	matplotlib スタイル・カラー・DPI・フォント・保存形式を設定
execute_python	Python コードを実行し図を保存、ANALYSIS_LOG に記録
build_report_tex	ANALYSIS_LOG から TeX 構築・tectonic でコンパイル
log_analysis_step	QIIME2 操作を ANALYSIS_LOG に手動登録

4.4 コードエージェントループの実装（vibe-local 方式）

code_agent.py の run_coding_agent() は「先に読む、後で書く」原則に基づくツール呼び出しループを実装する。LLM には **TOOL FIRST**（即座にツールを呼び出す）と **NEVER GIVE UP**（エラーが出ても修正して再実行）を指示しており、典型的な 1 解析タスクの流れは以下の通りである。

1. list_files でエクスポートファイルを把握
2. read_file で対象ファイルの列名・データ形式を確認
3. write_file で実際の列名を使った Python スクリプトを生成
4. run_python で実行し、exit code $\neq 0$ ならトレースバックを読んで write_file で修正し再実行

リスト 1 にループの中核部分を示す。

Listing 1: run_coding_agent() のツール呼び出しループ

```
def run_coding_agent(export_files, user_prompt, ...):
    messages = [{"role": "system", "content": SYSTEM_PROMPT},
                {"role": "user",   "content": task}]
    steps = 0
```

Table 2: code_agent.py のツール一覧（vibe-local 方式コード生成）

ツール名	機能
list_files	エクスポートディレクトリのファイル一覧とサイズを返す
read_file	ファイルの全内容を LLM に渡す。コード生成前に列名・データ形式を確認させることで精度を向上させる
write_file	mkstemp+replace による atomic write で Python スクリプトを安全に生成する
run_python	QIIME2 conda Python で実行；stdout・stderr・exit code を返す；新規生成図ファイルを自動検出する
install_package	ModuleNotFoundError を検出してユーザーに承認を求め、承認されれば pip install を実行する

```
    while steps < max_steps:
        response = call_ollama(messages, model, tools=
_TOOL_DEFS)
        tool_calls = response.get("tool_calls", [])

        if not tool_calls:
            break  # LLM が完了と判断

        for tc in tool_calls:
            fn   = tc["function"]
            name = fn["name"]
            args = fn["arguments"]   # dict or JSON 文字列
            if isinstance(args, str):
                args = json.loads(args)

            result, new_figs = _exec_tool(name, args, ...)
            figures.extend(new_figs)
            messages.append({"role": "tool",
                             "name": name,
                             "content": result[:4000]})
        steps += 1

    return CodeExecutionResult(success=bool(figures), figures=
figures)
```

4.5 小型モデル向けロバストネス機能

7B 以下の小型 LLM は Ollama の tool_calls フォーマットに非対応の場合があり、ツール呼び出しをプレーンテキストの JSON として出力する。seq2pipe はこれを run_coding_agent() 内の 4 層フォールバック機構で自動的に補う。

1. **テキストベースツール呼び出しパーサ（`_parse_text_tool_calls`）**: `tool_calls` が空のとき、応答本文を 5 つのヒューリスティックパターンで検索し JSON ツール呼び出しを抽出する: (a) "‘`json` コードブロック、(b) 応答全体を JSON として解析、(c) `"name": "..."` パターンのインライン正規表現スキャン、(d) name キーなし JSON をキー名から `write_file`/`list_files` として推論、(e) 壊れた JSON を正規表現で抽出する寛容なパーサ。

2. **Auto-inject `run_python`**: `write_file` が `.py` ファイルを書き込んだ直後、モデルが `run_python` を呼ぶのを待たずに自動実行する。これにより「スクリプトは書いたが実行を忘れる」という小型モデルの典型的な失敗を回避する。

3. **ステップ 6 フォールバック（1 ショット生成）**: `_run_python_count` カウンタが 5 ステップ後もゼロのとき（ループが進まない場合）、`run_coding_agent()` は自動的に `run_code_agent()` にフォールバックし、最大 3 回のエラー修正リトライ付き 1 ショット生成を行う。

4. **繰り返し検出（`call_ollama()` 内）**: 同じ 50 文字チャンクが 4 回連続して現れた場合、または応答が 20,000 文字を超えた場合に生成を強制終了し、無限ループを防ぐ。

これらの機構により、7B モデルでも少なくとも 1 種類以上の図を確実に生成できる。

4.6 QIIME2 パイプラインエージェントループ

`qiime2_agent.py` の `run_agent_loop()` は QIIME2 パイプライン生成の推論・行動サイクルを制御する。LLM の応答にツール呼び出しが含まれる場合は順次実行して結果を会話履歴に追加し、テキスト応答のみになるまで繰り返す。空レスポンスのガード処理も実装している。

4.7 Ollama API との通信と堅牢なエラー処理

`call_ollama()` 関数は Ollama の `/api/chat` エンドポイントに JSON リクエストを送信し、ストリーミングレスポンスを行単位で受信する。Python の `urllib.request` モジュールのみを使用しており、`requests` 等の外部パッケージに依存しない。

エラー処理として以下を実装した。

- `urllib.error.HTTPError`: Ollama サーバーの HTTP エラー（4xx/5xx）
- `urllib.error.URLError + socket.timeout`: タイムアウト（600 秒、環境変数 `SEQ2PIPE_PYTHON_TIMEOUT` で変更可）とサーバー未起動（`Connection refused`）を区別して表示
- `socket.timeout / TimeoutError`: ソケットレベルのタイムアウト

4.8 Python 下流解析と自動レポート生成

`tool_execute_python()` は Python コードを `subprocess.run()` 経由で実行する。実行前に `FIGURE_DIR`・`PLOT_CONFIG`・`FIGURE_FORMAT` 変数を自動注入するプリアンブルを付加するため、LLM はファイルパスを意識せずに解析コードを生成できる。実行結果のステップ・図パス・統計結果は `ANALYSIS_LOG` に自動蓄積される。

`tool_build_report_tex()` は `ANALYSIS_LOG` の内容から TeX ソースを Python で完全生成する（LLM を使用しない）。日本語版（XeLaTeX + xeCJK）と英語版（pdflatex）を別々に生成し、tectonic でコンパイルして PDF を出力する。

4.9 マニフェスト自動生成

`tool_generate_manifest()` は FASTQ ファイルの命名規則（`_R1_` / `_R2_` パターン）を正規表現（`re.sub(count=1)`）で解析し、ペアエンド・シングルエンド双方の QIIME2 マニフェスト TSV を自動生成する。R2 ファイルの探索は辞書ルックアップ（O(1)）で実装し大規模データセットにも対応する。Docker コンテナ内パス（デフォルト: `/data/output`）への変換も自動で行う。ペアが 1 組も見つからない場合はファイルを書かずにエラーを返す。

4.10 クロスプラットフォーム対応

macOS・Linux・Windows の差異を吸収するために以下の対策を実装した。

- **Docker 検出**: macOS では Docker Desktop の固定パス (`/Applications/Docker.app/.../dock` を優先し、その他の OS では `shutil.which("docker")` を使用する。
- **Windows ANSI カラー**: `os.system("")` を呼ぶことで Windows 10 以降における ANSI エスケープシーケンスを有効化する。
- **起動スクリプトの分離**: macOS/Linux 用（Bash）と Windows 用（PowerShell + バッチ）を別々に提供する。

5 解析ワークフロー

seq2pipe が生成する QIIME2 パイプラインは表 3 の 8 ステップで構成される。

Table 3: 生成される QIIME2 解析パイプラインの概要

STEP	処理	主要コマンド
1	データインポート	`qiime tools import`
2	クオリティ確認	`qiime demux summarize`
3	デノイジング（ASV 生成）	`qiime dada2 denoise-paired`
4	フィーチャーテーブル確認	`qiime feature-table summarize`
5	系統樹構築	`qiime phylogeny align-to-tree-mafft-fasttree`
6	分類学的解析	`qiime feature-classifier classify-sklearn`
7	多様性解析	`qiime diversity core-metrics-phylogenetic`
8	差次解析（オプション）	`qiime composition ancombc`

SILVA 138 参照データベース [5] を用いた Naive Bayes 分類器により、16S rRNA 増幅領域の分類学的同定を行う。増幅領域（V1–V3: 27F/338R, V3–V4: 341F/806R, V4: 515F/806R）に応じてプライマートリム長・トランケーション長が自動的に調整される。

QIIME2 解析後、自律エージェントモード（`-auto`）では以下の 5 フェーズ・**14 種類の図**を全自動で生成する。コードエージェントは各ファイルを `read_file` で先に読んで列名・形式を確認してからコードを生成するため、フォーマットエラーが起きにくい。

1. **Phase 0 (quality)**: デノイジング統計（入力 / フィルタリング / デノイジング / 非キメラ）
2. **Phase 1 (alpha)**: Shannon / Chao1 / Simpson / Faith's PD の計算・可視化・Mann-Whitney U + Kruskal-Wallis 検定
3. **Phase 2 (beta)**: Bray-Curtis PCoA・UniFrac PCoA・CLR 変換 PCA（組成データ向け主成分分析）・NMDS（非計量多次元尺度構成法）・全 多様性 TSV ファイルを使ったレアファクション曲線
4. **Phase 3 (taxonomy)**: 門レベル stacked bar・属レベル heatmap・CLR 変換 phylum bar
5. **Phase 4 (correlation)**: サンプル間相関行列 + 階層クラスタリング heatmap

全フェーズ完了後、`build_report_tex` が `ANALYSIS_LOG` から日英 TeX/PDF レポートを自動生成する。

6 対応モデルと性能比較

seq2pipe は Ollama で動作する任意の LLM と組み合わせて使用できる。表 4 に推奨モデルとその特性を示す。

Table 4: 対応モデル一覧

モデル	必要 RAM	モデルサイズ	特徴
`qwen2.5-coder:7b`	8 GB 以上	約 4.7 GB	コード生成特化（推奨）
`qwen2.5-coder:3b`	4 GB 以上	約 1.9 GB	軽量・高速
`llama3.2:3b`	4 GB 以上	約 2.0 GB	汎用・対話能力高め
`qwen3:8b`	16 GB 以上	約 5.2 GB	最高品質・推論能力高い

7 考察と今後の課題

7.1 達成された目標

seq2pipe は以下の目標を達成した。

- Python 標準ライブラリのみを用いたゼロ依存実装
- ローカル LLM による完全オフライン動作
- 3 OS（macOS / Linux / Windows）への完全対応
- FASTQ データ構造の自動認識と適応的パイプライン生成

- QIIME2 コマンドを conda 環境 / Docker 経由で安全に実行するコマンド確認機構
- **vibe-local 方式ツール呼び出し型コード生成エージェント**: LLM がファイルを先に読んでから Python コードを生成するため、盲目的な 1 ショット生成で頻発したフォーマット不一致エラーを根本解消
- **自動エラー修正（NEVER GIVE UP）**: `run_python` が失敗するとエージェントがトレースバックを読んでコードを修正し、`EXIT CODE: 0` になるまで再実行
- **5 フェーズ・14 種類の図を自動生成**: クオリティ確認・多様性（4 指標）・多様性（PCoA + CLR-PCA + NMDS + レアファクション）・分類組成（3 図）・サンプル相関
- **ModuleNotFoundError 自動復旧**: 不足パッケージを検出してユーザーに承認を求め、`pip install` を実行
- **小型 LLM 向けロバストネス機能**: 4 層フォールバック機構(`_parse_text_tool_calls`・Auto-inject `run_python`・ステップ 6 の 1 ショットフォールバック・繰り返し検出）により、7B モデルでも確実に図を生成
- 2 つの操作モード: 指定解析（モード 1）・完全自律（モード 2 / `-auto`）
- `ANALYSIS_LOG` からの日英 TeX/PDF レポート自動生成（LLM 不使用）
- 日本語 / 英語の起動時言語選択 UI
- 接続エラー・タイムアウト・空レスポンス・ファイルシステムエラーへの堅牢な処理

7.2 現在の制限

- LLM の生成するコマンドの正確性はモデルに依存し、誤ったパラメータが生成される可能性がある。
- 大規模データセット（100 サンプル以上）に対するパフォーマンスは未検証である。
- SILVA 138 参照ファイルのダウンロードには約 30 GB のディスク容量と数時間を要する。
- TeX レポートのコンパイルには tectonic のインストールが必要である。
- `qiime2_agent.py`（QIIME2 パイプライン生成）には read-first パターンが未適用であり、`code_agent.py` との統合が今後の課題である。

7.3 今後の課題

- Web UI の提供（Streamlit GUI —`app.py` として実装予定）
- ITS / 18S rRNA など他のマーカー遺伝子への対応
- パイプライン実行結果のフィードバックループ実装

- 並列ツール実行による処理の高速化
- 差次存在量解析（volcano plot）・機械学習（Random Forest）の自律タスクへの追加
- 統計的検定結果の図への自動アノテーション

8 おわりに

本稿では、ローカル LLM と QIIME2 を統合した自動解析エージェント seq2pipe の設計・実装を報告した。本システムは 2 つの相補的な LLM エージェントを組み合わせる: `qiime2_agent.py` が 11 ツールで QIIME2 パイプライン生成を担い、`code_agent.py` が vibe-local 方式の 5 ツールで実際のファイル内容を読んでから Python 解析コードを生成し、エラーが出ても自動修正して最終的に成功させる。これにより研究者は生 FASTQ データから 5 フェーズ・14 種類の出版品質の図と日英 PDF レポートまでを、完全オフラインかつクラウド不要の環境で自動化できる。

seq2pipe はオープンソース(MIT ライセンス)として公開されており、`https://github.com/Rhizo` からアクセスできる。

8 References

[1] Bolyen, E., et al. (2019). *Reproducible, interactive, scalable and extensible microbiome data science using QIIME 2*. Nature Biotechnology, 37, 852–857.

[2] Brown, T. B., et al. (2020). *Language Models are Few-Shot Learners*. Advances in Neural Information Processing Systems, 33, 1877–1901.

[3] Yao, S., et al. (2023). *ReAct: Synergizing Reasoning and Acting in Language Models*. International Conference on Learning Representations (ICLR 2023).

[4] Ollama (2023). *Ollama: Get up and running with large language models locally*. https://ollama.com/

[5] Quast, C., et al. (2013). *The SILVA ribosomal RNA gene database project: improved data processing and web-based tools*. Nucleic Acids Research, 41(D1), D590–D596.

[6] Callahan, B. J., et al. (2016). *DADA2: High-resolution sample inference from Illumina amplicon data*. Nature Methods, 13(7), 581–583.

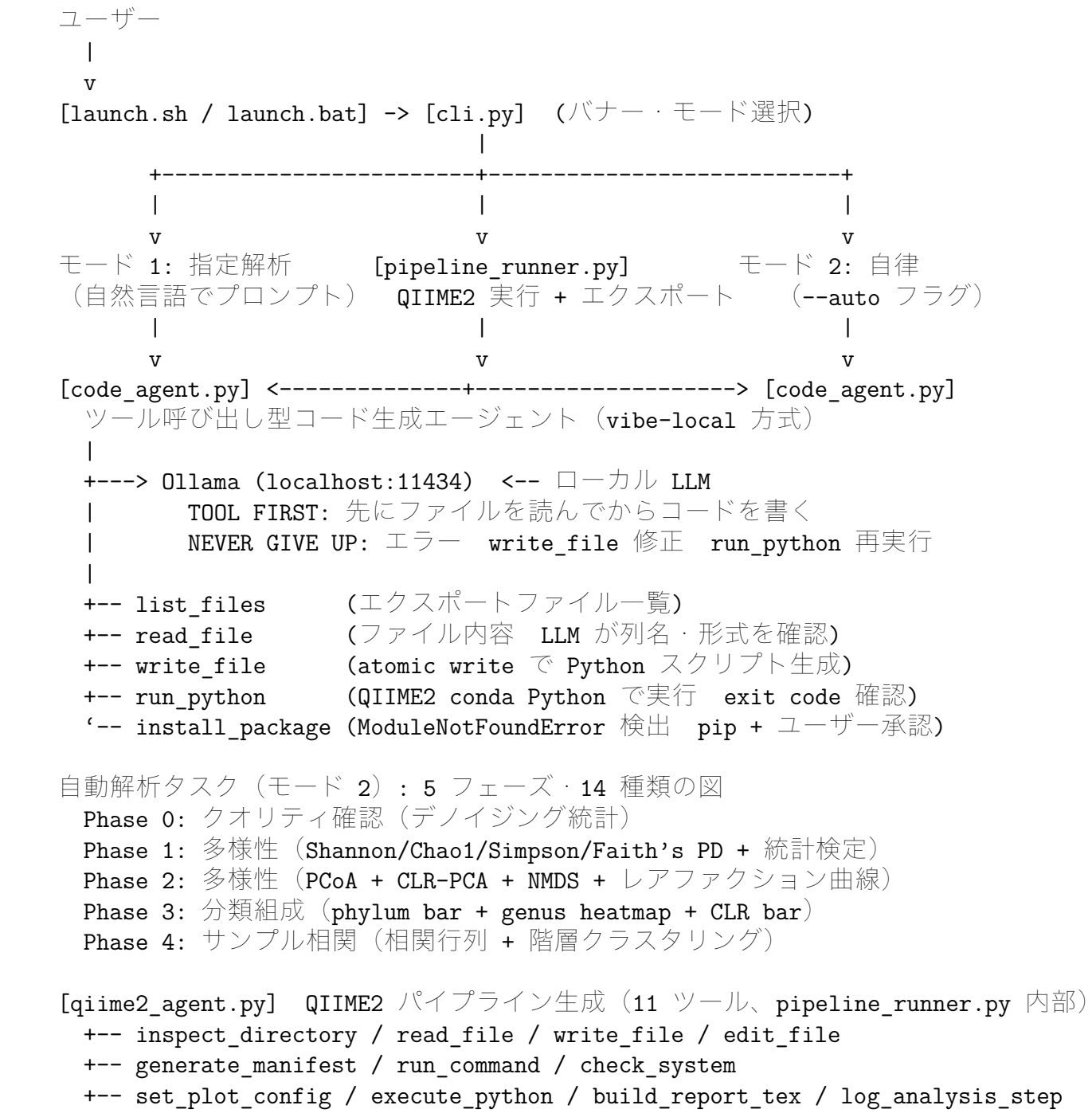

Figure 1: seq2pipe の 3 層アーキテクチャ