seq2pipe

A Local LLM-Powered AI Agent for Automated QIIME2 Pipeline Generation

Rhizobium-gits Claude (Anthropic)

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

February 23, 2026

Abstract

We present **seq2pipe**, an interactive AI agent that automates end-to-end microbiome analysis by combining a locally running large language model (LLM) with the QIIME2 bioinformatics platform. Given raw FASTQ sequencing data, seq2pipe automatically inspects the data structure, designs an appropriate QIIME2 analysis pipeline, generates ready-to-execute shell scripts, and then invokes a second **tool-calling code-generation agent** (`code_agent.py`) that reads actual exported file contents before writing Python code — eliminating format-mismatch errors common in blind one-shot generation. The code agent follows a "NEVER GIVE UP" policy: on any Python error it rewrites and retries until `EXIT CODE: 0`. In autonomous mode it automatically produces 14 publication-quality figures across 5 analysis phases (quality control, alpha diversity, beta diversity including CLR-PCA and NMDS, taxonomic composition, and sample correlation). The entire workflow runs on the user's local machine, eliminating dependencies on cloud services or paid APIs. A startup language selector (Japanese/English), automatic Python dependency checking, and comprehensive error handling make seq2pipe immediately usable by researchers without bioinformatics backgrounds.

1 Introduction

QIIME2 [1] (Quantitative Insights Into Microbial Ecology 2) has become the de facto standard platform for 16S rRNA amplicon sequencing analysis in microbiome research. However, QIIME2 carries a steep learning curve: users must understand multiple data formats, select appropriate parameters for denoising algorithms (DADA2), manage Docker-containerized execution environments, and interpret multi-step analysis outputs. These barriers make QIIME2 inaccessible to many researchers, particularly those without bioinformatics backgrounds. Furthermore, visualization of QIIME2 outputs (`.qzv` artifacts) typically requires an online viewer (`view.qiime2.org`), creating an additional dependency that is unavailable in offline or restricted-network environments.

The rapid advancement of large language models (LLMs) has opened the door to domain-specific analysis automation through natural language [2]. Yet cloud-hosted LLM services introduce concerns about cost, data privacy, and network availability. The emergence of local LLM inference frameworks such as Ollama [4] allows high-quality language reasoning to be performed entirely on commodity hardware, resolving these concerns.

In this report, we describe the design, architecture, and implementation of **seq2pipe**, an interactive AI agent that integrates local LLM inference with QIIME2 to provide a fully automated, offline-capable microbiome analysis assistant. seq2pipe goes well beyond pipeline generation: it performs Python-based downstream statistical analysis, saves publication-quality figures, and automatically generates bilingual research reports — all within a single conversational interface.

2 Background and Related Work

2.1 Complexity of QIIME2 Analysis

A complete QIIME2 microbiome analysis involves at least eight major steps: data import, quality inspection, denoising (ASV generation via DADA2), feature table summarization, phylogenetic tree construction, taxonomic classification (using SILVA [5]), diversity analysis, and differential abundance analysis. Each step requires careful selection of command-line parameters that depend on the sequencing library configuration (paired-end vs. single-end), the 16S rRNA hypervariable region amplified (V1–V3, V3–V4, V4, etc.), and primer sequences.

2.2 LLM Agents and Function Calling

The ReAct framework [3] established the paradigm of LLMs that interleave reasoning and action — calling external tools at appropriate times to gather information or perform operations that cannot be accomplished through text generation alone. Modern instruction-tuned LLMs supporting function calling can select and invoke pre-defined tools with structured arguments, enabling autonomous task completion.

2.3 Local LLM Inference with Ollama

Ollama wraps the `llama.cpp` inference engine into a single binary distributable across macOS, Linux, and Windows. It exposes a REST API (`/api/chat`) compatible with the OpenAI Chat Completions format, including support for function calling, making it an ideal backend for locally deployed AI agents.

3 System Design

3.1 Design Principles

seq2pipe was developed according to seven core design principles:

1. **Zero external dependencies**: Only Python's standard library (`json`, `urllib`, `subprocess`, `pathlib`, `socket`, etc.) is used, eliminating the need for `pip install`.
2. **Fully local execution**: Ollama's local inference engine is used exclusively, requiring no internet connection after initial setup.
3. **Cross-platform compatibility**: The agent runs on macOS, Linux, and Windows.
4. **Safe command execution**: All shell commands require explicit user confirmation before execution.

5. **Embedded domain knowledge**: The system prompt encodes a complete QIIME2 workflow knowledge base.

6. **Multilingual UI**: The user selects Japanese or English at startup; all AI responses and auto-generated reports follow this preference.

7. **Robust error handling**: Connection errors, timeouts, empty responses, and filesystem errors are all caught and reported with user-friendly messages.

3.2 Overall Architecture

Figure 1 illustrates the three-layer architecture of seq2pipe. The user launches `cli.py` via platform-specific launch scripts. `cli.py` presents a rainbow ASCII banner and prompts the user to select between two operating modes. In either mode, `pipeline_runner.py` first executes the full QIIME2 pipeline and exports the results, then `code_agent.py` — a vibe-local-style tool-calling agent — performs Python downstream analysis.

4 Implementation Details

4.1 Startup Sequence and Multilingual UI

On launch, `select_language()` prompts the user to choose Japanese (`ja`) or English (`en`). The selection is stored in the `LANG` global variable, and the `ui()` function returns all interface text in the chosen language. Auto-generated reports follow the same language setting.

`check_python_deps()` verifies that numpy, pandas, matplotlib, and seaborn are available at startup via `subprocess`, guiding the user to install any missing packages before analysis begins.

4.2 Embedding Domain Knowledge in the System Prompt

The `SYSTEM_PROMPT` variable contains a comprehensive QIIME2 workflow knowledge base. This includes:

- Automatic data format detection criteria (e.g., paired-end detection via `*_R1*.fastq.gz` patterns)
- Complete QIIME2 commands for all eight analysis steps (import through differential abundance analysis)
- Region-specific recommended parameters for V1–V3 (27F/338R), V3–V4 (341F/806R), and V4 (515F/806R) amplicons
- Docker execution command templates
- Metadata file format specifications
- SILVA 138 taxonomic hierarchy explanation
- Common errors and troubleshooting guidance
- Python downstream analysis and autonomous exploration mode guidelines

By embedding this knowledge directly into the system prompt, a general-purpose code LLM is transformed into a QIIME2 domain expert.

4.3 Tool Definitions and Function Calling

seq2pipe uses two separate tool sets. `qiime2_agent.py` defines 11 tools (Table 1) for QIIME2 pipeline orchestration. `code_agent.py` defines 5 tools (Table 2) for Python code generation following the vibe-local paradigm: the LLM *reads the actual file contents before writing code*, eliminating format-mismatch errors.

Table 1: Tools in `qiime2_agent.py` (QIIME2 pipeline orchestration)

Tool Name	**Function**
`inspect_directory`	Lists files with sizes; auto-identifies FASTQ, QZA, metadata
`read_file`	Reads text files (TSV, CSV, Markdown, etc.) up to 50 lines
`write_file`	Writes scripts, manifests, and README files
`edit_file`	Replaces a unique string in an existing file
`generate_manifest`	Auto-generates QIIME2 manifest TSV from a FASTQ directory
`run_command`	Executes shell commands after user confirmation; auto-detects QIIME2 conda bin, supports `SEQ2PIPE_AUTO_YES`
`check_system`	Verifies Docker, Ollama, Python, and disk space
`set_plot_config`	Configures matplotlib style, palette, DPI, and output format
`execute_python`	Executes Python analysis code; saves figures to `FIGURE_DIR`; logs steps to `ANALYSIS_LOG`
`build_report_tex`	Builds TeX/PDF reports from `ANALYSIS_LOG` via tectonic
`log_analysis_step`	Manually registers QIIME2 steps into `ANALYSIS_LOG`

4.4 The Code Agent Loop (vibe-local Style)

`run_coding_agent()` in `code_agent.py` implements a "read-first, code-second" tool-calling loop. The LLM is instructed to **always call a tool immediately** (TOOL FIRST principle) and **never give up on errors** (NEVER GIVE UP principle). The typical sequence per analysis task is:

1. `list_files` — discover available exported files
2. `read_file` — read a target file to inspect column names and data format

Table 2: Tools in `code_agent.py` (vibe-local style code generation)

Tool Name	Function
`list_files`	Lists all files in the exported results directory with their sizes
`read_file`	Returns the full content of a file (TSV, CSV, Python, etc.) to the LLM, allowing it to inspect column names and data format *before* writing code
`write_file`	Writes a Python script atomically (via `mkstemp`+`replace`) to prevent partial writes
`run_python`	Executes the written script using the QIIME2 conda Python interpreter; returns stdout, stderr, and exit code; detects new figure files
`install_package`	Detects `ModuleNotFoundError`, prompts user for approval, then runs `pip install`

3. `write_file` — generate a Python script that uses the exact column names seen in step 2
4. `run_python` — execute the script; if exit code $\neq 0$, the LLM reads the traceback and loops back to `write_file` to fix the error

Listing 1 shows the core loop structure.

Listing 1: Core tool-calling loop in run_coding_agent()

```
def run_coding_agent(export_files, user_prompt, ...):
    messages = [{"role": "system", "content": SYSTEM_PROMPT},
                {"role": "user",   "content": task}]
    steps = 0
    while steps < max_steps:
        response = call_ollama(messages, model, tools=
   _TOOL_DEFS)
        tool_calls = response.get("tool_calls", [])

        if not tool_calls:
            break  # LLM says it's done

        for tc in tool_calls:
            fn   = tc["function"]
            name = fn["name"]
            args = fn["arguments"]  # dict or JSON string
            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 Robustness Enhancements for Small LLMs

Smaller models (7B parameters and below) often do not emit structured `tool_calls` objects via the Ollama API; instead, they embed JSON tool invocations as plain text in the response body. seq2pipe handles this through a four-layer fallback system implemented in `run_coding_agent()`:

1. **Text-based tool call parser (`_parse_text_tool_calls`)**: When `tool_calls` is empty, the response body is searched for JSON objects using five heuristic patterns: (a) `"‘json` code blocks, (b) the entire response as a JSON object, (c) inline regex scan for `"name": "..."` patterns, (d) name-less JSON objects inferred as `write_file`/`list_files` by key inspection, and (e) a lenient regex-based broken-JSON extractor.

2. **Auto-inject `run_python`**: Immediately after `write_file` successfully writes a `.py` file, `run_python` is automatically executed without waiting for the LLM to call it — avoiding the common failure mode where small models write a script but forget to run it.

3. **Step-6 fallback to 1-shot generation**: A `_run_python_count` counter tracks how many times `run_python` has been called (both via tool calls and auto-inject). If no execution has occurred after 5 steps, `run_coding_agent()` falls back to `run_code_agent()`, which performs 1-shot code generation with up to 3 error-correction retries — ensuring figure output even when the tool-calling loop stalls.

4. **Repetition detector in `call_ollama()`**: A sliding-window check truncates generation when the same 50-character chunk appears four consecutive times, or when total output exceeds 20 000 characters — preventing infinite loops caused by degenerate model outputs.

Together, these mechanisms guarantee that at least one analysis figure is produced regardless of model size or Ollama version.

4.6 The QIIME2 Pipeline Agent Loop

`run_agent_loop()` in `qiime2_agent.py` controls the QIIME2 pipeline generation cycle. It sends conversation history to the LLM, detects tool calls, executes them sequentially, appends results, and repeats until the LLM produces a text-only response. An empty-response guard retries automatically if the LLM returns neither content nor tool calls.

4.7 Communication with the Ollama API and Error Handling

The `call_ollama()` function sends JSON requests to Ollama's `/api/chat` endpoint with streaming enabled, and processes server-sent events line by line using only `urllib.request` from Python's standard library.

Comprehensive error handling covers:

- `urllib.error.HTTPError`: Ollama server HTTP errors (4xx/5xx)
- `urllib.error.URLError + socket.timeout`: distinguishes timeouts (300 s) from connection-refused errors (Ollama not running)
- `socket.timeout / TimeoutError`: socket-level timeout fallback

4.8 Python Downstream Analysis and Automatic Report Generation

`tool_execute_python()` executes Python code via `subprocess.run()` with a timeout of 300 seconds. Before execution, a preamble is prepended that injects `FIGURE_DIR`, `PLOT_CONFIG`, and `FIGURE_FORMAT` variables, allowing the LLM to generate analysis code without knowing the exact file paths. Each execution appends a structured entry (step name, figure paths, statistical results) to the global `ANALYSIS_LOG`.

`tool_build_report_tex()` reads `ANALYSIS_LOG` and programmatically generates complete TeX source — no LLM involvement required, making it fast and deterministic. Separate Japanese (XeLaTeX + xeCJK) and English (pdflatex) documents are produced and compiled to PDF via tectonic.

4.9 Automatic Manifest Generation

The `tool_generate_manifest()` function uses regular expressions (`re.sub(count=1)`) to parse FASTQ filenames, automatically generating QIIME2 manifest TSV files for both paired-end and single-end data. R2 file lookup is implemented with a dictionary for O(1) performance, scaling to large datasets. If no R1/R2 pairs are found, the function returns an error without writing an empty manifest. Partial matching rates below 80% trigger an enhanced warning showing the mismatch percentage. Path translation to Docker container-internal paths (default: `/data/output`) is performed automatically.

4.10 Cross-Platform Compatibility

Three platform-specific differences are handled explicitly:

- **Docker detection**: On macOS, the Docker Desktop binary path (`/Applications/Docker.app/.` is checked first; on other platforms, `shutil.which("docker")` is used.
- **Windows ANSI color support**: Calling `os.system("")` activates ANSI escape sequence processing in Windows 10+ terminals.
- **Separated launch scripts**: Bash shell scripts for macOS/Linux and PowerShell + batch files for Windows are provided separately.

5 Analysis Workflow

The QIIME2 pipeline generated by seq2pipe consists of eight steps as summarized in Table 3.

Taxonomic classification is performed using a Naive Bayes classifier trained on the SILVA 138 reference database [5]. Primer trim lengths and truncation positions are automatically adjusted based on the detected hypervariable region (V1–V3: 27F/338R, V3–V4: 341F/806R, V4: 515F/806R).

Table 3: Overview of the generated QIIME2 analysis pipeline

Step	Process	Key Command
1	Data import	`qiime tools import`
2	Quality visualization	`qiime demux summarize`
3	Denoising (ASV generation)	`qiime dada2 denoise-paired`
4	Feature table summarization	`qiime feature-table summarize`
5	Phylogenetic tree construction	`qiime phylogeny align-to-tree-mafft-fasttree`
6	Taxonomic classification	`qiime feature-classifier classify-sklearn`
7	Diversity analysis	`qiime diversity core-metrics-phylogenetic`
8	Differential abundance (opt.)	`qiime composition ancombc`

Following QIIME2 analysis, the autonomous agent mode (`-auto`) executes five phases of Python downstream analysis, producing **14 publication-quality figures**. Unlike simple script injection, the code agent first calls `read_file` on each target TSV to confirm exact column names, then writes and executes analysis code.

1. **Phase 0 (quality)**: Denoising statistics — input, filtered, denoised, non-chimeric read counts
2. **Phase 1 (alpha diversity)**: Shannon / Chao1 / Simpson / Faith's PD with Mann-Whitney U and Kruskal-Wallis tests
3. **Phase 2 (beta diversity)**: Bray-Curtis PCoA, UniFrac PCoA, CLR-transformed PCA (principal component analysis on compositional data), NMDS (non-metric multidimensional scaling), and rarefaction curves across all available beta-diversity TSV files
4. **Phase 3 (taxonomy)**: Phylum-level stacked bar chart (relative abundance), genus-level heatmap with hierarchical clustering, and CLR-transformed phylum bar chart
5. **Phase 4 (sample correlation)**: Pairwise sample correlation matrix with hierarchical clustering heatmap

Upon completion, `build_report_tex` automatically generates bilingual TeX/PDF reports from `ANALYSIS_LOG`.

6 Supported Models and Performance

seq2pipe is model-agnostic and can be used with any LLM available in Ollama. Table 4 lists recommended models.

The code-generation-optimized `qwen2.5-coder:7b` is recommended as the default model. For RAM-constrained environments (e.g., 8 GB MacBook Air), `qwen2.5-coder:3b` or `llama3.2:3b` provide a lighter-weight alternative with minimal capability trade-off for structured QIIME2 command generation.

Table 4: Supported models

Model	RAM Required	Size	Characteristics
`qwen2.5-coder:7b`	8 GB+	~4.7 GB	Code generation optimized (recommended)
`qwen2.5-coder:3b`	4 GB+	~1.9 GB	Lightweight and fast
`llama3.2:3b`	4 GB+	~2.0 GB	General purpose, strong dialogue
`qwen3:8b`	16 GB+	~5.2 GB	Highest quality, strong reasoning

7 Setup and Launch

7.1 Software Requirements

Table 5 summarizes the required software stack.

Table 5: Software requirements

Software	Purpose	Installation
Python 3.9+	Agent runtime	Typically pre-installed
Ollama	Local LLM inference	Automated via `setup.sh`
Docker Desktop/Engine	QIIME2 execution environment	Manual install
numpy, pandas, etc.	Python downstream analysis	Automated via `setup.sh`
tectonic (optional)	PDF report compilation	`brew install tectonic`

7.2 Launch Sequence

1. Run `setup.sh` (macOS/Linux) or `setup.bat` (Windows) to install Ollama, download the LLM model, and install Python packages.
2. Run `launch.sh` / `launch.bat`: the script verifies Ollama and Docker availability, then starts the agent.
3. Select Japanese or English at the language prompt.
4. Through natural language dialogue, provide the data directory path and specify the desired analyses (taxonomic composition, diversity, differential abundance, etc.).
5. The agent automatically inspects the data, generates a customized script bundle, runs Python analyses, and produces a PDF report.

8 Discussion

8.1 Achieved Goals

seq2pipe achieves the following:

- Zero-dependency implementation using only Python's standard library
- Fully offline operation via local LLM inference

- Complete cross-platform support (macOS / Linux / Windows)
- Automatic recognition of FASTQ data structure and adaptive pipeline generation
- Safe QIIME2 command execution with explicit user confirmation
- **Vibe-local style tool-calling code agent** that reads actual file contents before writing code, eliminating format-mismatch errors that plagued blind one-shot generation approaches
- **Automated error correction**: when `run_python` fails, the agent reads the traceback, rewrites the script, and retries until `EXIT CODE: 0` (NEVER GIVE UP policy)
- **5-phase autonomous analysis producing 14 figures**: quality control, alpha diversity (4 metrics), beta diversity (PCoA + CLR-PCA + NMDS + rarefaction curves), taxonomy (3 chart types), and sample correlation
- **ModuleNotFoundError auto-recovery**: detects missing packages, prompts user for approval, and runs `pip install`
- **Small-LLM robustness**: four-layer fallback system (`_parse_text_tool_calls`, auto-inject `run_python`, step-6 1-shot fallback, repetition detector) ensures reliable figure generation even with 7B parameter models
- Two operation modes: directed (Mode 1 natural language prompt) and fully autonomous (Mode 2 `-auto` flag)
- Automatic bilingual (Japanese/English) TeX/PDF report generation from `ANALYSIS_LOG` without LLM involvement
- Startup language selection UI (Japanese / English)
- Robust handling of connection errors, timeouts, empty responses, and filesystem errors

8.2 Current Limitations

- The accuracy of generated QIIME2 commands depends on the underlying LLM model quality; incorrect parameters may be produced.
- Performance with large datasets (100+ samples) has not been formally evaluated.
- Initial SILVA 138 classifier training requires approximately 30 GB of disk space and several hours of computation.
- PDF report compilation requires tectonic to be installed separately.
- The QIIME2 pipeline agent (`qiime2_agent.py`) does not yet apply the read-first pattern; integrating it with `code_agent.py`'s approach is a future goal.

8.3 Future Directions

- Web UI integration (Gradio or Streamlit) — a Streamlit prototype (`app.py`) is already planned
- Support for additional marker genes (ITS, 18S rRNA)
- Feedback loop incorporating pipeline execution results to iteratively refine parameters
- Parallel tool execution for faster autonomous exploration
- Integration of statistical result annotation directly into figures
- Extending the autonomous task list with differential abundance (volcano plots) and machine learning (Random Forest) analyses

9 Conclusion

We have described seq2pipe, an interactive AI agent that integrates local LLM inference with the QIIME2 microbiome analysis platform. The system combines two complementary LLM agents: `qiime2_agent.py` orchestrates 11 specialized tools for QIIME2 pipeline generation, while `code_agent.py` — a vibe-local-style tool-calling agent with 5 tools — generates accurate Python analysis code by reading actual file contents before writing code and automatically correcting errors until success. Together they enable researchers to automate an entire 16S rRNA analysis workflow — from raw FASTQ data through 14 publication-quality figures across 5 analysis phases, and bilingual research reports — entirely offline and without any cloud dependencies.

seq2pipe is released as open source under the MIT License and is available at: `https://github.com/l`

9 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.

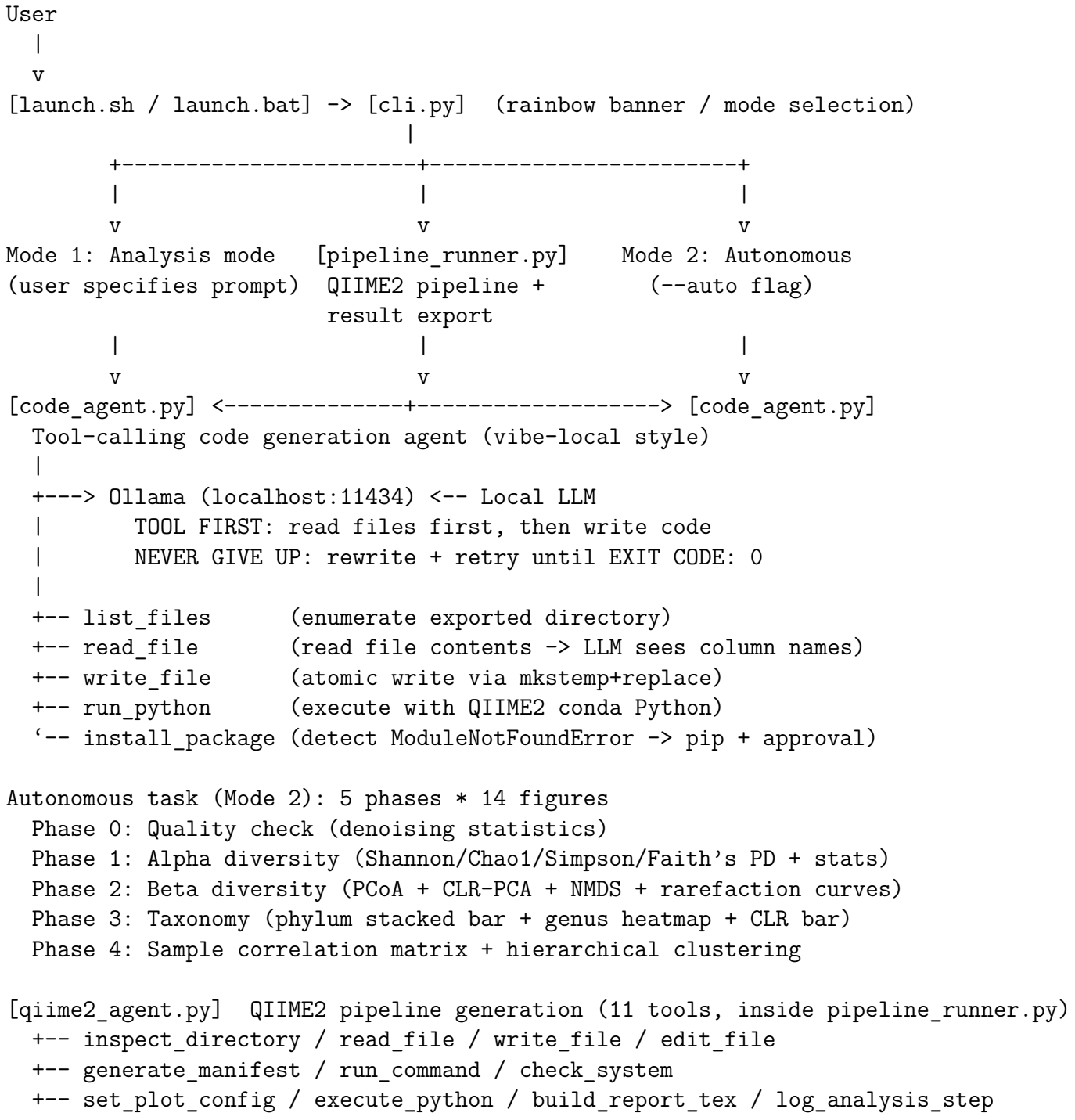

Figure 1: Three-layer architecture of seq2pipe