seq2pipe

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

Rhizobium-gits Claude (Anthropic)

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

February 28, 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 **deterministic comprehensive analysis module** (`analysis.py`) that generates **15 publication-quality PNG figures without any LLM dependency**. Additionally, a **tool-calling code-generation agent** (`code_agent.py`) 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 (`-auto`), the system completes all analysis in 3 steps: STEP 1 (QIIME2 pipeline) → STEP 2 (deterministic analysis, 15 figures) → STEP 3 (HTML report). The `-classifier` option enables SILVA 138 taxonomic classification, automatically generating genus/phylum composition figures (fig13–fig15). After analysis completes, a **post-analysis refinement mode** is activated where users can iteratively refine figures through natural language instructions (e.g. "change colors", "move legend outside"). The entire workflow runs on the user's local machine, eliminating dependencies on cloud services or paid APIs.

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: its deterministic comprehensive analysis module reliably produces 15 publication-quality figures, the LLM agent provides flexible additional analysis, and the system 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 eight 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.
8. **Guaranteed deterministic analysis**: An LLM-independent comprehensive analysis module (`analysis.py`) reliably generates 15 figures regardless of model quality.

3.2 Overall Architecture

Figure 1 illustrates the 3-step 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 `-auto` mode, a 3-step automated pipeline executes: STEP 1 runs `pipeline_runner.py` for the QIIME2 pipeline and exports results, STEP 2 runs `analysis.py` to deterministically generate 15 PNG figures, and STEP 3 auto-generates an HTML report. In Mode 1 (directed analysis), `code_agent.py` provides vibe-local-style tool-calling code generation.

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

```
User
  |
  v
[launch.sh] -> [cli.py]  (rainbow banner / mode selection)
                   |
  +----------------+---------------------------+
  |                |                           |
  v                v                           v
Mode 1      [pipeline_runner.py]        Mode 2
Analysis     QIIME2 pipeline +           --auto flag
mode         result export
  |                |   (STEP 1/3)              |
  |                v                           |
  |       [analysis.py]  (STEP 2/3)            |
  |         Deterministic analysis (no LLM)    |
  |         15 PNG figures reliably generated   |
  |         fig01-12: Core analysis             |
  |         fig13-15: Taxonomy (with classifier)|
  |                |                           |
  v                v                           v
[code_agent.py] <-+----> [report_generator.py]
  Tool-calling           HTML / LaTeX+PDF
  code generation        report generation
  agent                    (STEP 3/3)
  |
  +---> Ollama (localhost:11434)
```

Figure 1: Three-step architecture of seq2pipe

- 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 lualatex
`log_analysis_step`	Manually registers QIIME2 steps into `ANALYSIS_LOG`

4.4 Deterministic Comprehensive Analysis Module (`analysis.py`)

To improve the reliability of `-auto` mode, we introduced a deterministic comprehensive analysis module `analysis.py` that operates entirely without LLM dependency. The `run_comprehensive_analysis()` function reads QIIME2 exported data directly and uses pandas, matplotlib, seaborn, and scikit-learn to generate 15 publication-quality PNG figures (Table 3).

Because this module does not depend on LLM response quality, identical analysis results are produced regardless of which model is used. When SILVA 138 classifier results are available, fig13–fig15 (genus/phylum composition) are automatically generated.

4.5 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()

```python
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,
```

Table 3: 15 figures generated by `analysis.py`

Figure	Analysis	Key Packages
fig01	DADA2 denoising statistics	pandas, matplotlib
fig02	Sequencing depth per sample	pandas, matplotlib
fig03	Alpha diversity boxplots	pandas, seaborn
fig04	Shannon diversity per sample	pandas, seaborn
fig05	PCoA (Bray-Curtis)	sklearn MDS
fig06	PCoA (Jaccard)	sklearn MDS
fig07	PCoA (Unweighted UniFrac)	sklearn MDS
fig08	PCoA (Weighted UniFrac)	sklearn MDS
fig09	Beta diversity heatmaps (2×2)	seaborn
fig10	Top 30 ASV heatmap	seaborn
fig11	Alpha diversity correlations	matplotlib
fig12	ASV richness vs depth	matplotlib
fig13	Genus-level stacked bar*	matplotlib
fig14	Phylum-level stacked bar*	matplotlib
fig15	Genus-level heatmap*	seaborn

* Generated only when SILVA 138 classifier results are available

```
                                   "content": result[:4000]})
        steps += 1

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

4.6 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. Furthermore, since `analysis.py` runs before the LLM agent, 15 core figures are always generated regardless of LLM performance.

4.7 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.

4.8 SILVA 138 Classifier Auto-Discovery (`-classifier`)

The `-classifier` option was added to `cli.py` to specify the path to a SILVA 138 Naive Bayes classifier. When no path is explicitly given, `_find_classifier()` searches candidate locations: `~/seq2pipe/silva-138-99-nb-classifier.qza`, `~/classifiers/` subdirectory, and `/usr/local/share/qiime2/` directory. When found, QIIME2 performs taxonomic classification, and `analysis.py` automatically generates genus/phylum composition figures.

4.9 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

4.10 Automatic PNG Conversion for Figures

When the LLM generates `plt.savefig()` calls with `.pdf` or `.svg` extensions, the resulting files cannot be opened in macOS Preview. The `_convert_new_figs()` helper automatically detects such files immediately after code execution and converts them to PNG using the macOS built-in `sips` command, then removes the original. The system prompt was also strengthened with an explicit instruction that the extension "MUST be `.png`", reducing the frequency of incorrect extension choices from the LLM.

4.11 Diversity Analysis Without Metadata

The standard `qiime diversity core-metrics-phylogenetic` command requires `-m-metadata-file` as a mandatory argument; when no metadata file is provided, the entire diversity analysis step (Step 7) was previously skipped. This was corrected by adding a fallback branch that individually invokes `qiime diversity alpha`, `qiime diversity beta`, `qiime diversity alpha-phylogenetic`, and `qiime diversity beta-phylogenetic` for each metric (Shannon, Faith's PD, Bray-Curtis, UniFrac, etc.) when metadata is absent.

4.12 Post-Analysis Refinement Mode and Report Generation

After analysis completes, `_run_refinement_session()` launches an interactive loop where users can issue natural-language instructions to refine the generated figures. Each instruction is passed to `run_refinement_loop()`, which has the LLM modify the existing `analysis.py` script and re-execute it. The loop exits on empty Enter or `quit`.

In `-auto` mode, an HTML report is automatically generated as STEP 3. Additionally, within the refinement session, keyword detection triggers report generation:

- "report" / "html" → `generate_html_report()`: a self-contained HTML file with figures embedded as base64 data URIs, LLM-generated Japanese captions per figure, an overall summary, and a parameter table.
- "PDF" / "latex" → `generate_latex_report()`: auto-detects `lualatex` (preferred, using `luatexja-preset` for Japanese) or `xelatex` (fallback, using `xeCJK` + Hiragino fonts on macOS). Compiles twice for cross-reference resolution. If no LaTeX engine is found, `report.tex` is saved for manual compilation.

4.13 Python Downstream Analysis

`tool_execute_python()` executes Python code via `subprocess.run()` with a configurable timeout. Before execution, a preamble is prepended that injects `FIGURE_DIR`, `DPI`, and required imports, allowing the LLM to generate analysis code without knowing exact paths. Generated figures are detected by comparing the directory listing before and after execution, and PDF/SVG files are immediately converted to PNG.

4.14 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.

4.15 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 4.

Table 4: 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`

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).

Following QIIME2 analysis, the `-auto` mode executes a 3-step automated pipeline:

1. **STEP 1**: QIIME2 pipeline (8 steps above) + result export
2. **STEP 2**: Deterministic comprehensive analysis (`analysis.py`) → **15 PNG figures**:
   - fig01: DADA2 denoising statistics
   - fig02: Sequencing depth per sample
   - fig03: Alpha diversity (Shannon / Faith PD / Observed ASVs)
   - fig04: Shannon diversity per sample
   - fig05–08: PCoA (Bray-Curtis / Jaccard / Unweighted UniFrac / Weighted UniFrac)
   - fig09: Beta diversity distance heatmaps (2×2)
   - fig10: Top 30 ASV heatmap
   - fig11: Alpha diversity correlations
   - fig12: ASV richness vs depth

  - fig13: Genus-level composition (with classifier)
  - fig14: Phylum-level composition (with classifier)
  - fig15: Genus-level heatmap (with classifier)

3. **STEP 3**: Automatic HTML report generation (base64-embedded figures)

Upon completion, the post-analysis refinement mode activates, allowing natural-language figure refinement and additional report generation (`report_generator.py`).

6 Supported Models and Performance

seq2pipe is model-agnostic and can be used with any LLM available in Ollama. Table 5 lists recommended models.

Table 5: 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

Note that the deterministic analysis (STEP 2) via `analysis.py` does not use the LLM, so all 15 figures are generated identically regardless of model choice. Model quality primarily affects the flexibility of Mode 1 (directed analysis) and the quality of the refinement mode.

7 Setup and Launch

7.1 Software Requirements

Table 6 summarizes the required software stack.

Table 6: 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`
lualatex / xelatex (optional)	PDF report compilation	`brew install -cask mactex-no-gu`

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 report.

8 Example Results

To demonstrate seq2pipe's end-to-end capability, we analysed 10 human stool samples (TEST01–TEST10, freeze-dried, Illumina MiSeq paired-end V3–V4) without manual intervention. The full pipeline ran in a single command:

Listing 2: Fully autonomous analysis with seq2pipe

```
./launch.sh --fastq-dir ~/input --auto --threads 1
```

STEP 1 executed the QIIME2 pipeline (DADA2 denoising + phylogenetic tree + diversity analysis + SILVA 138 taxonomic classification), STEP 2 generated 15 PNG figures via `analysis.py`, and STEP 3 produced an HTML report automatically. Figures are stored in the `Figure/` directory of this repository.

8.1 DADA2 Denoising Statistics

The progression of reads through each DADA2 stage (input → filtered → denoised → merged → non-chimeric) is shown in Figure 2.

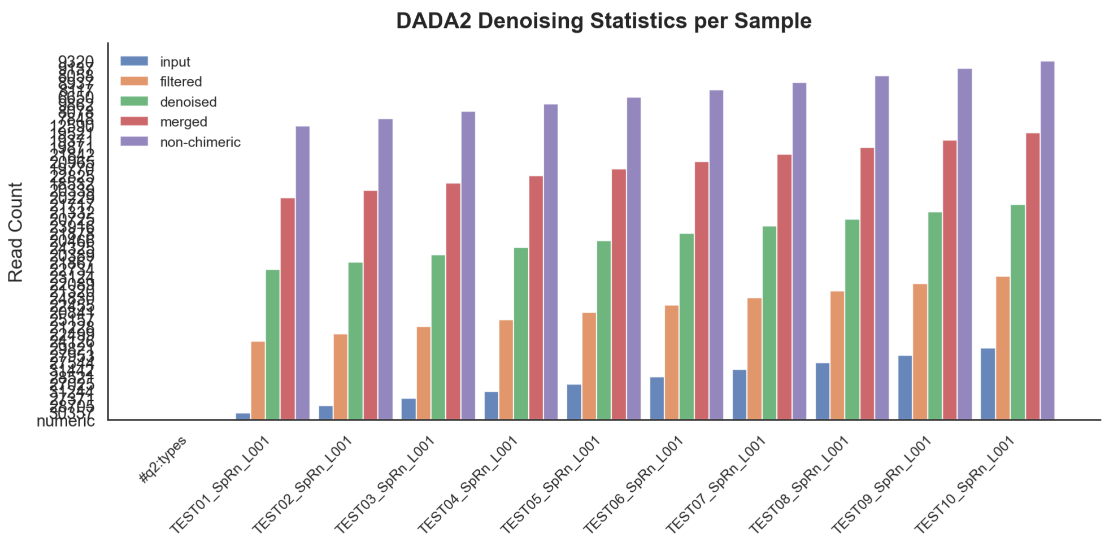

Figure 2: DADA2 denoising statistics for 10 stool samples. The figure reveals consistent filtering efficiency across samples.

8.2 Alpha Diversity

Three alpha diversity metrics (Shannon entropy, Faith's PD, and observed ASVs) were computed per sample and displayed as boxplots (Figure 3). A per-sample Shannon diversity strip plot is shown in Figure 4.

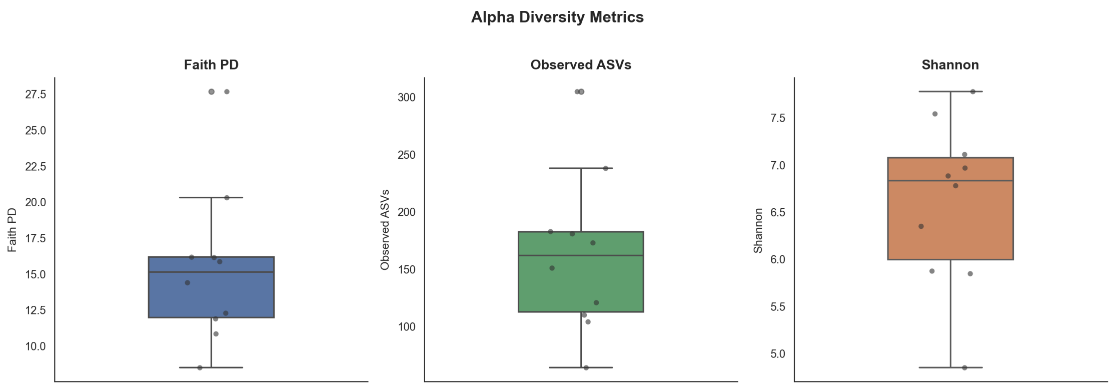

Figure 3: Alpha diversity of 10 human stool samples. Each panel shows a different metric computed from the DADA2-denoised feature table.

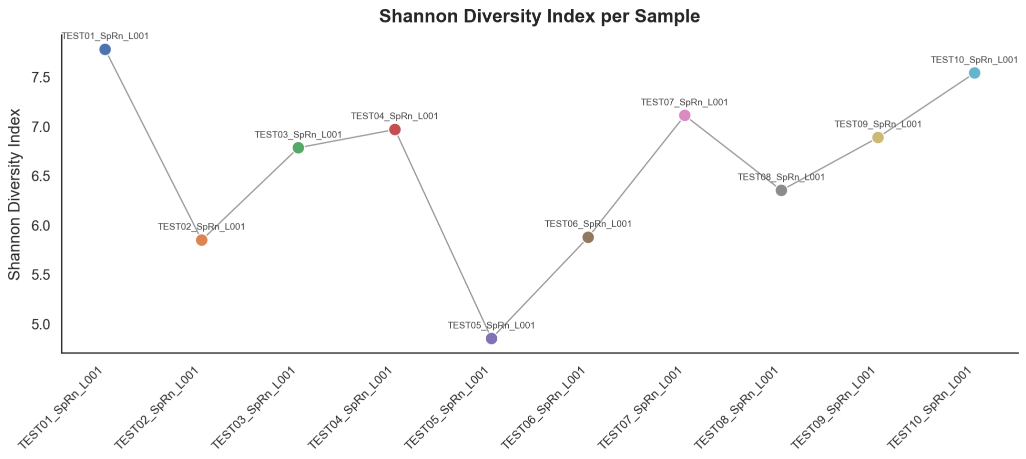

Figure 4: Shannon alpha diversity per sample (strip plot).

8.3 Beta Diversity

Beta diversity was assessed using four dissimilarity measures. Bray–Curtis PCoA (Figure 5) and Unweighted UniFrac PCoA (Figure 6) were computed via scikit-learn MDS (`n_components=2, dissimilarity='precomputed'`). A 2×2 panel of distance heatmaps is shown in Figure 7.

8.4 Taxonomic Composition

Using SILVA 138 classifier results, genus-level stacked bar charts (Figure 8) and genus-level heatmaps (Figure 9) were generated.

All 15 figures and the HTML report were generated automatically by `analysis.py` and `report_generator.py`.

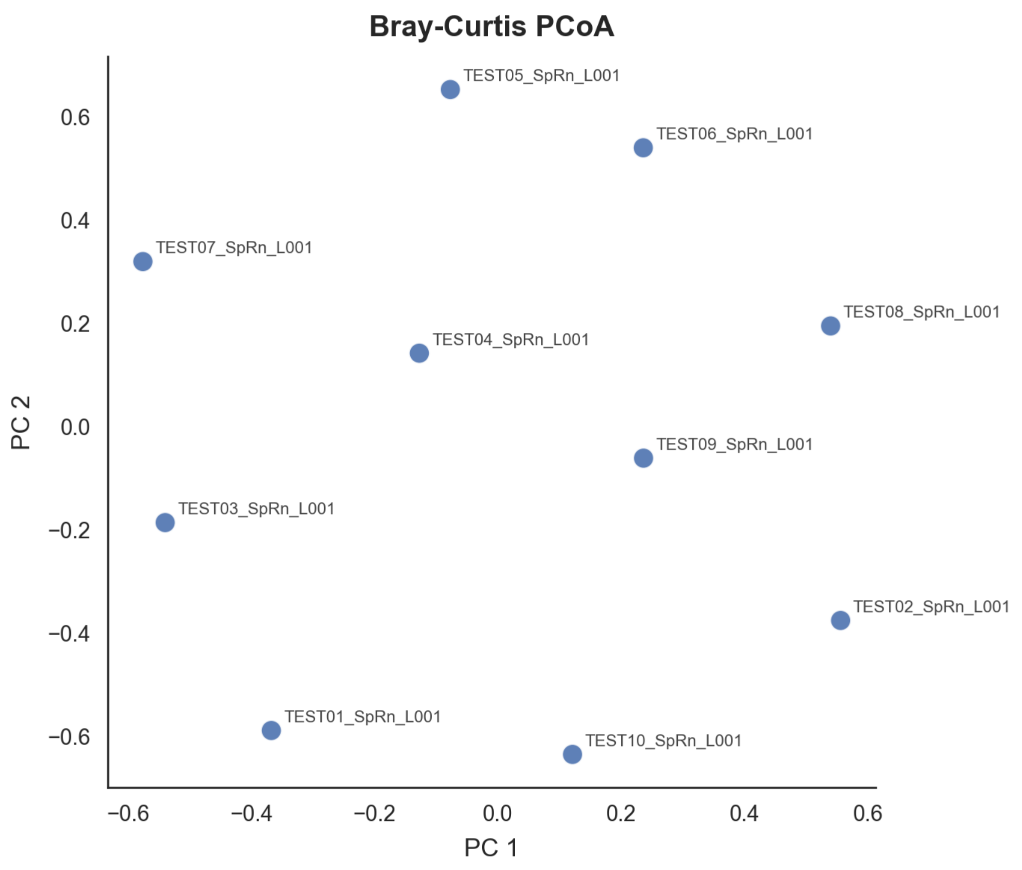

Figure 5: Bray–Curtis PCoA. Samples are coloured by ID and labelled directly on the plot.

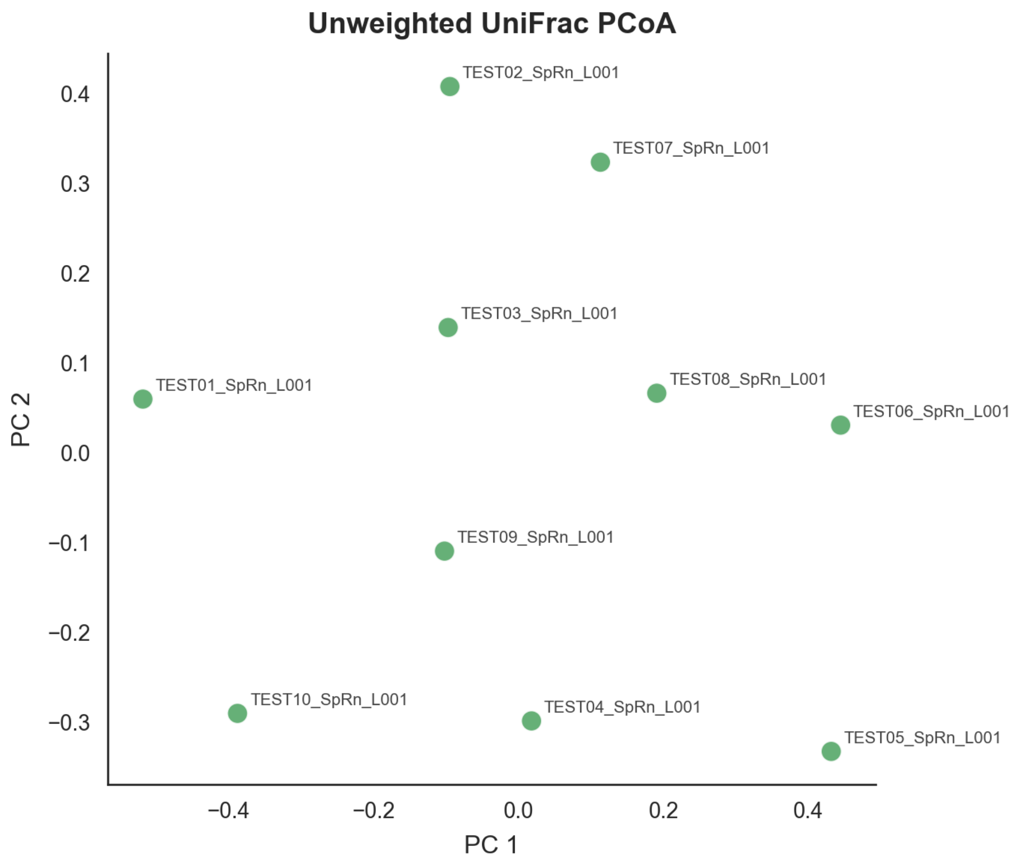

Figure 6: Unweighted UniFrac PCoA. Phylogenetic distances computed by QIIME2 are projected onto two MDS dimensions.

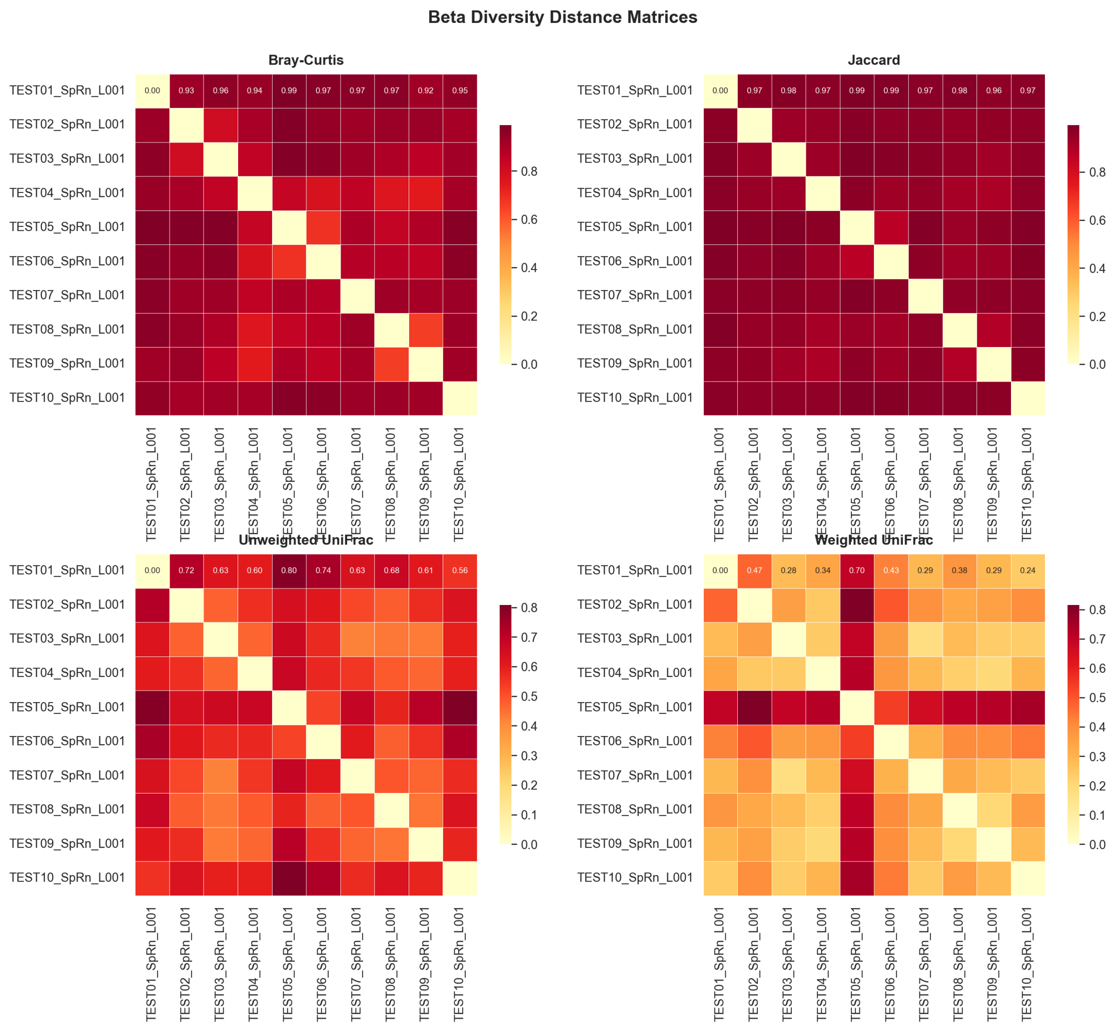

Figure 7: Beta diversity distance heatmaps (2×2): Bray-Curtis, Jaccard, Unweighted UniFrac, and Weighted UniFrac.

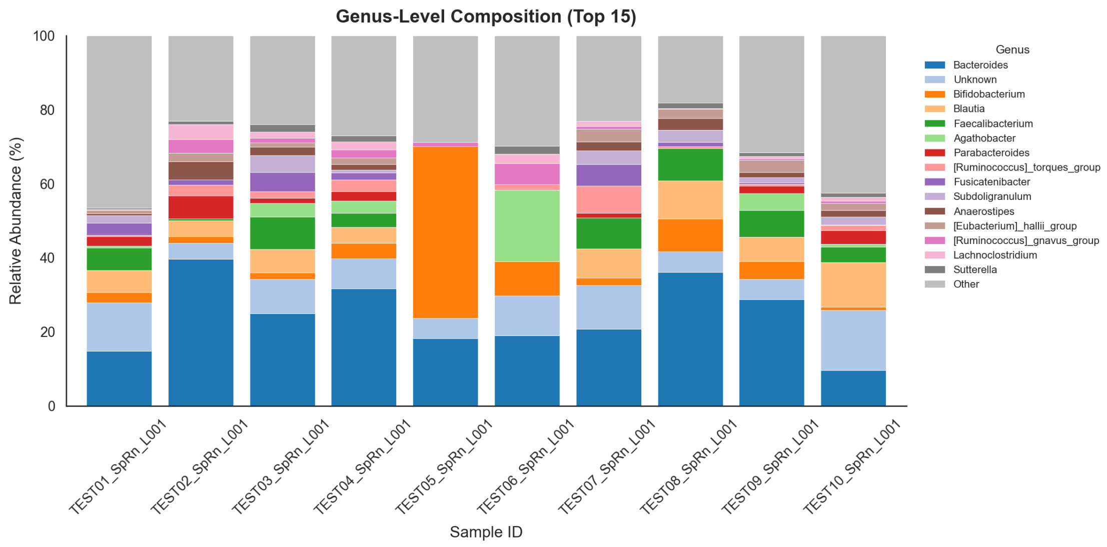

Figure 8: Genus-level taxonomic composition (stacked bar chart). Top genera are colour-coded to show microbial community structure across samples.

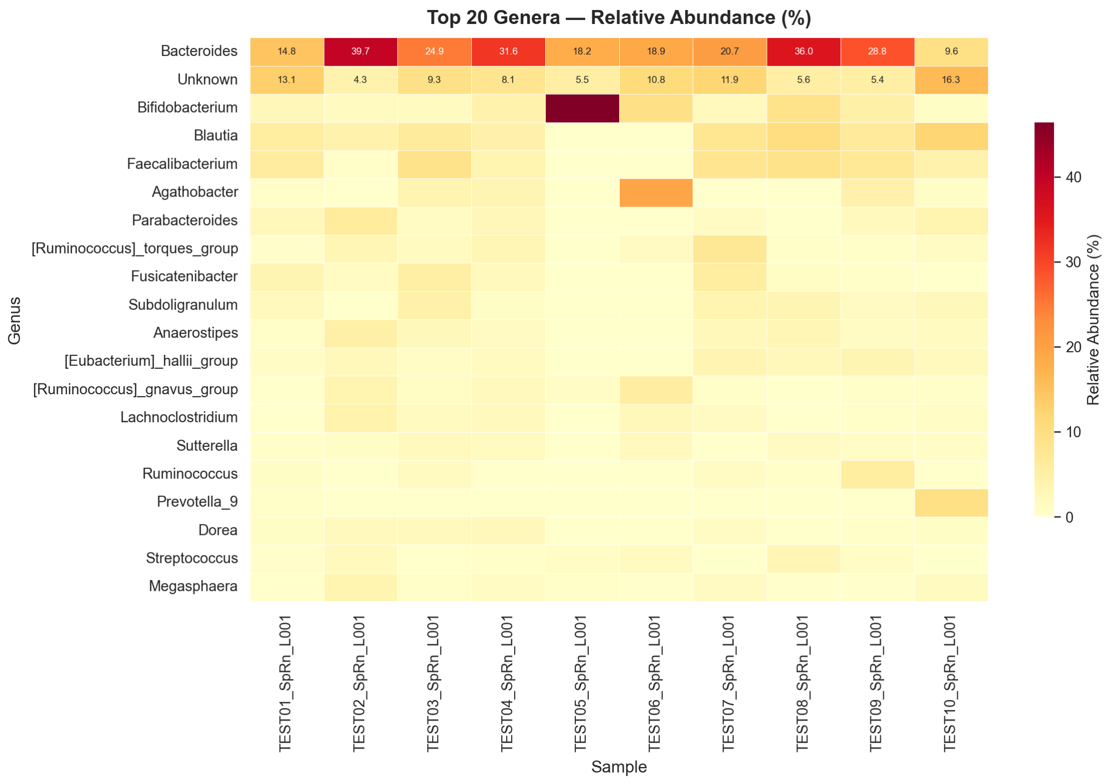

Figure 9: Genus-level heatmap showing relative abundances across samples.

9 Discussion

9.1 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
- **Deterministic comprehensive analysis module (`analysis.py`)**: 15 publication-quality PNG figures generated reliably without LLM dependency
- **SILVA 138 classifier auto-discovery (`-classifier`)**: automatic genus/phylum composition figure generation
- **3-step automated pipeline**: STEP 1 (QIIME2) → STEP 2 (deterministic analysis) → STEP 3 (HTML report)
- **Vibe-local style tool-calling code agent** that reads actual file contents before writing code, eliminating format-mismatch errors

- **Automated error correction**: NEVER GIVE UP policy — rewrites and retries until `EXIT CODE: 0`
- **Small-LLM robustness**: four-layer fallback system
- Two operation modes: directed (Mode 1) and fully autonomous (Mode 2)
- **Post-analysis refinement mode**: iterative figure refinement via natural language
- **Automatic PDF/SVG to PNG conversion (`sips`)**: no extra dependencies on macOS
- **Metadata-free diversity analysis**: individual commands as fallback when metadata is absent
- **HTML and LaTeX/PDF report auto-generation**
- Startup language selection UI (Japanese / English)

9.2 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.
- PDF report compilation requires `lualatex` or `xelatex` (MacTeX / TeX Live); if unavailable, `report.tex` is saved for manual compilation.
- PDF/SVG auto-conversion relies on macOS `sips`; Linux/Windows users must convert manually (Pillow-based conversion is planned).

9.3 Future Directions

- Support for additional marker genes (ITS, 18S rRNA)
- Pillow-based PDF/SVG $\rightarrow$ PNG conversion for Linux and Windows
- Extending the autonomous task list with differential abundance (volcano plots) and machine learning (Random Forest) analyses
- Integration of statistical result annotation directly into figures
- Adding rarefaction curves, NMDS, and other analyses to `analysis.py`

10 Conclusion

We have described seq2pipe, an interactive AI agent that integrates local LLM inference with the QIIME2 microbiome analysis platform. The system combines three complementary modules: `qiime2_agent.py` orchestrates 11 specialized tools for QIIME2 pipeline generation, `analysis.py` deterministically generates 15 publication-quality PNG figures without LLM dependency, and `code_agent.py` — a vibe-local-style tool-calling agent with 5 tools — provides flexible additional analysis by reading actual file contents before writing code and automatically correcting errors. A post-analysis refinement mode allows iterative figure improvement via natural language, and `report_generator.py` produces HTML or LaTeX/PDF reports. Together they enable researchers to automate an entire 16S rRNA analysis workflow — from raw FASTQ data through 15 publication-quality figures and an HTML report — entirely offline and without any cloud dependencies.

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

10 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.