seq2pipe

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

Rhizobium-gits Claude (Anthropic)

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

2025

Abstract

We present **seq2pipe**, an interactive AI agent that automates 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 produces a visualization guide — all through natural language dialogue. The entire processing pipeline runs on the user's local machine, eliminating dependencies on cloud services or paid APIs. The implementation uses only Python's standard library and leverages Ollama's local LLM inference engine to support function calling. Evaluation with the `qwen2.5-coder:7b` model demonstrates that practical, well-parameterized QIIME2 pipelines can be reliably generated.

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.

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.

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 five core design principles:

1. **Zero external dependencies**: Only Python's standard library (`json`, `urllib`, `subprocess`, `pathlib`, 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.

3.2 Overall Architecture

Figure 1 illustrates the overall architecture of seq2pipe. The user launches the Python agent (`qiime2_agent.py`) via platform-specific launch scripts. The agent communicates with the local Ollama API endpoint (`http://localhost:11434/api/chat`) and coordinates six specialized tools to interact with the filesystem and QIIME2 (via Docker).

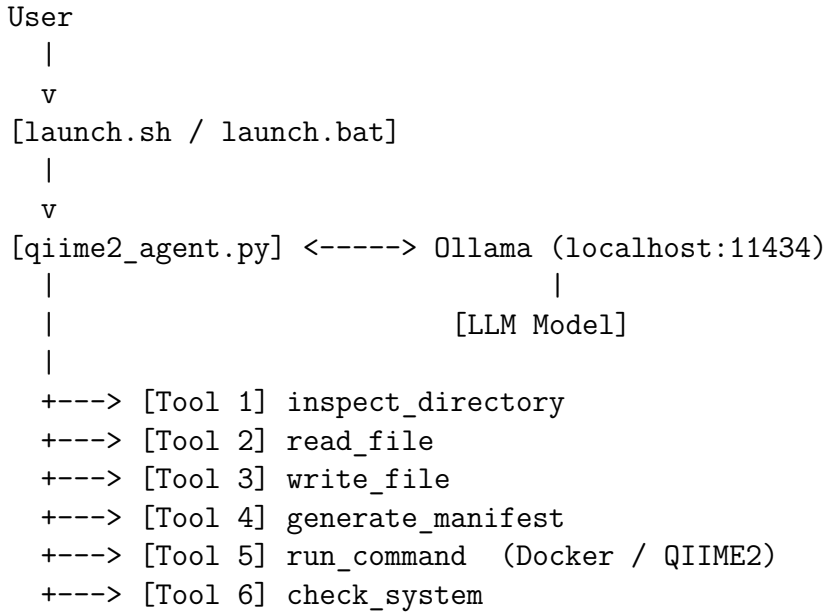

Figure 1: Overall architecture of seq2pipe

4 Implementation Details

4.1 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

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

4.2 Tool Definitions and Function Calling

Six tools are defined in JSON Schema format compatible with Ollama's function calling interface. Table 1 summarizes the tools and their roles.

Table 1: Tools provided by seq2pipe

Tool Name	Function
`inspect_directory`	Lists files in a directory with sizes and automatically identifies FASTQ, QZA, and metadata files
`read_file`	Reads text files (TSV, CSV, Markdown, etc.) up to 50 lines
`check_system`	Verifies installation status and versions of Docker, Ollama, and Python
`write_file`	Writes generated scripts, manifests, and README files to disk
`generate_manifest`	Auto-generates a QIIME2 manifest TSV from a FASTQ directory
`run_command`	Executes shell commands after explicit user confirmation

4.3 The Agent Loop

The `run_agent_loop()` function controls the core reasoning-action cycle. It sends conversation history to the LLM, detects tool calls in the response, executes them sequentially, appends results to the conversation history, and repeats until the LLM produces a final text-only response (Listing 1).

Listing 1: Core agent loop implementation

```
def run_agent_loop(messages: list, model: str):
    while True:
        response = call_ollama(messages, model, tools=TOOLS)
        assistant_msg = {"role": "assistant",
                         "content": response["content"]}

        if response["tool_calls"]:
            tool_results = []
            for tc in response["tool_calls"]:
                fn = tc.get("function", {})
                result = dispatch_tool(fn["name"],
                                       fn.get("arguments", {}))
                tool_results.append({"role": "tool",
                                     "content": result})
            assistant_msg["tool_calls"] = response["tool_calls"
  ]
```

```
            messages.append(assistant_msg)
            messages.extend(tool_results)
            continue    # Query LLM again with tool results
        else:
            messages.append(assistant_msg)
            break       # Text-only response: end loop
```

4.4 Communication with the Ollama API

The `call_ollama()` function sends JSON requests to Ollama's `/api/chat` endpoint with streaming enabled, and processes server-sent events line by line. It uses only `urllib.request` from Python's standard library, without any third-party HTTP clients.

Listing 2: Ollama API communication

```
def call_ollama(messages, model, tools=None):
    body = {"model": model, "messages": messages,
            "stream": True}
    if tools:
        body["tools"] = tools
    data = json.dumps(body).encode("utf-8")
    req = urllib.request.Request(
        OLLAMA_URL, data=data,
        headers={"Content-Type": "application/json"},
        method="POST"
    )
    with urllib.request.urlopen(req, timeout=300) as resp:
        for raw_line in resp:
            chunk = json.loads(raw_line.decode())
            # Process streaming output and tool call detection
            ...
```

4.5 Cross-Platform Compatibility

Three platform-specific differences are handled explicitly:

- **Docker detection**: On macOS, the Docker Desktop binary path (`/Applications/Docker.app/C` is checked first; on other platforms, `shutil.which("docker")` is used (`_get_docker_cmd()`).
- **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.

4.6 Automatic Manifest Generation

The `tool_generate_manifest()` function uses regular expressions to parse FASTQ filenames (`_R1_` / `_R2_` patterns), automatically generating QIIME2 manifest TSV files for both paired-end and single-end data. It also performs automatic path translation to Docker container-internal paths (default: `/data/output`).

5 Analysis Workflow

The QIIME2 pipeline generated by seq2pipe consists of eight steps as summarized in Table 2.

Table 2: 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).

6 Supported Models and Performance

seq2pipe is model-agnostic and can be used with any LLM available in Ollama. Table 3 lists recommended models.

Table 3: 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

The code-generation-optimized `qwen2.5-coder:7b` is recommended as the default model. For RAM-constrained environments, `qwen2.5-coder:3b` or `llama3.2:3b` provide a lighter-weight alternative.

7 Setup and Launch

7.1 Software Requirements

Table 4 summarizes the required software stack.

Table 4: 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

7.2 Launch Sequence

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

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

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.

8.3 Future Directions

- Implementation of automated command validation and error correction
- Web UI integration (Gradio or Streamlit)
- Support for additional marker genes (ITS, 18S rRNA)
- Feedback loop incorporating pipeline execution results to iteratively refine parameters

9 Conclusion

We have described seq2pipe, an interactive AI agent that integrates local LLM inference with the QIIME2 microbiome analysis platform. By embedding comprehensive QIIME2 domain knowledge into the system prompt and providing six specialized tools through Ollama's function calling interface, seq2pipe enables researchers to automate complex 16S rRNA analysis workflows through natural language dialogue — 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.