fleet-rlm: One-Page App Summary

Evidence sources: README.md, AGENTS.md, docs/explanation/architecture.md, pyproject.toml, src/fleet_rlm/*.py

What it is

fleet-rlm is a Python package for Recursive Language Models (RLM) that combines DSPy planning with Modal sandbox execution for secure long-context code workflows. It lets an agent generate Python code, run it in an isolated cloud container, and use results iteratively to answer complex tasks.

Who it is for

Primary persona: AI/agent developers (including Claude Code users) who need safe, programmatic analysis of large documents or datasets without local execution risk.

What it does

- Runs RLM tasks where DSPy plans and generates code while Modal executes that code in an isolated sandbox.
- Provides Typer CLI commands for basic Q&A, architecture/API extraction, error pattern analysis, and trajectory/tool demos.
- Offers OpenTUI-based interactive chat (`fleet-rlm code-chat --opentui`) with WebSocket-backed interaction.
- Exposes an optional FastAPI server with health, chat, task, and WebSocket routes.
- Exposes an optional MCP server (`fleet-rlm serve-mcp`) for tool-oriented integrations.
- Supports stateful execution, helper tooling, and sub-LLM tool calls (`llm_query`) via interpreter/driver protocol.

How it works (architecture overview)

- Entry surfaces: CLI (`src/fleet_rlm/cli.py`) and optional FastAPI app (`src/fleet_rlm/server/main.py`).
- Orchestration layer: runner functions and `RLMReActChatAgent` build DSPy tasks and drive iterative reasoning.
- Execution layer: `ModalInterpreter` starts/controls a Modal sandbox and communicates with sandbox driver over JSON via stdio.
- Sandbox layer: `src/fleet_rlm/core/driver.py` executes generated Python with persistent state and helper functions, then streams structured outputs back.
- Data flow: user request -> planner/code generation -> sandbox execution/observations -> iterative refinement -> final response to CLI/API/TUI.

How to run (minimal getting started)

- From repo root: `uv sync --extra dev`
- Create local env file: `cp .env.example .env`
- Configure Modal once: `uv run modal setup`
- Create secret (replace values): `uv run modal secret create LITELLM DSPY_LM_MODEL=... DSPY_LM_API_BASE=... DSPY_LLM_API_KEY=...`
- Run a first command: `uv run fleet-rlm run-basic --question "What are the first 12 Fibonacci numbers?"`
- Not found in repo: exact required model/provider values for `LITELLM` secret fields.