# RLM Algorithm: Implementation vs. Paper Comparison

Comparison of our `rlm-loop` implementation against the original MIT CSAIL paper (arXiv:2512.24601).

---

## Our Algorithm (as implemented)

```
User Query + Context (file/dir/string)


1. Materialize context → plain Python str
2. Optionally build "context sample" (--no-context-sample disables)
3. Create depth-tracked llm_query(snippet, task) closure (max_depth=3)
4. Initialize REPLEnv with namespace: CONTEXT, FILES, llm_query, FINAL,
   SHOW_VARS, pre-imported modules


5. Send to LLM: system prompt (full/compact) + user query [+ context sample]


   Iteration Loop (max_iterations, timeout, token budget)

    6. Check timeout / token budget guards
    7. LLM responds with markdown
    8. Extract ```python blocks (regex)
    9. If no code blocks:
         - Check for "FINAL" in text → done
         - Else prompt LLM to write code
   10. Execute each code block sequentially
         in persistent REPL namespace
   11. If FINAL() called → done
   12. Feed captured print() output back
         as user message: "Output:\n..."
   13. Continue loop




Return RLMResult(answer, stats, history)
```

Key files: `rlm/rlm.py` (orchestrator), `rlm/repl.py` (REPL environment), `rlm/prompts.py` (system prompts), `rlm/backends.py` (LLM backends)

---

## Paper's Algorithm

```
User Query + Context


1. Store context as CONTEXT variable in Python REPL
2. Provide llm_query(snippet, task) for recursive sub-LM calls
3. Root LM receives ONLY the query (context is not inline)


4. Send to LLM: system prompt + query (no context sample)


   Code-Observe-Reason Loop

  5. LLM generates Python code
  6. Execute in REPL sandbox
  7. Observe output (truncated)
  8. LLM decides next action:
       - More code (inspect/search/chunk)
       - llm_query() for semantic analysis
       - FINAL(answer) when confident
  9. Continue loop




Return final answer
```

---

## Side-by-Side Comparison

| Aspect | Paper | Our Implementation | Delta |
|---|---|---|---|
| **Initial context exposure** | Root LM sees only the query; CONTEXT is opaque until code runs | Context sample included by default; `--no-context-sample` disables it | **Aligned** — can now match paper's opaque-context design via flag |
| **`llm_query` signature** | `llm_query(snippet, task)` — two args | `llm_query(snippet, task)` — two args | **Match** |

| Aspect | Paper | Our Implementation | Delta |
|---|---|---|---|
| **Recursive depth** | Depth-1 only (root → sub-LM, no deeper) | Configurable `max_depth=3` (supports multi-level recursion) | We're ahead of the paper here |
| **Sub-LM model** | Smaller/cheaper model for sub-calls (e.g., GPT-5-mini when root is GPT-5) | `--recursive-model` param exists but defaults to same model as `--model` | Paper's design is more cost-efficient |
| **System prompt for sub-calls** | Not detailed | No system prompt — bare user message only | Consistent but sub-calls might benefit from minimal guidance |
| **Termination** | `FINAL()` + max iterations + timeouts | `FINAL()` + max iterations + `--timeout` + `--max-token-budget` | **Match** |
| **Default model** | Not applicable (paper uses specific models per experiment) | `--model` is required (no defaults) | **Aligned** — user must choose explicitly |
| **Code extraction** | Not specified in detail | Regex: `` ```python\n(.*?)``` `` — only explicitly tagged Python blocks | Reasonable; ignores non-Python fenced blocks |
| **Sandbox isolation** | Rootless container (security delegated to runtime) | Safe builtins whitelist (no `__import__`, `open`, `exec`) + container delegation | We add in-process restrictions on top of container isolation |
| **Output truncation** | Mentioned (first ~10K chars) | 10,000 char cap per execution | Match |

| Aspect | Paper | Our Implementation | Delta |
|---|---|---|---|
| **Multi-file support** | Not discussed | `FILES` dict + `CompositeContext` for directories | Extension beyond the paper |
| **Context loading** | Context as string | Memory-mapped files via `LazyContext`, `CompositeContext` for multi-file | Engineering refinement beyond the paper |
| **Async execution** | Identified as future work | Stubs exist (`acompletion`) but delegate to sync | Neither implements true async |
| **Native post-training** | RLM-Qwen3-8B (28.3% improvement over base) | Prompt-only, no fine-tuning | Paper has a training component we don't |
| **Cost tracking** | Discussed (comparable median, high variance tail) | `RLMStats` tracks tokens, iterations, recursive calls; `--max-token-budget` enforces limits | **Aligned** — cost enforcement now available |
| **Verification strategy** | Some models perform redundant verification sub-calls | Not prompted for; up to the LLM | Paper observes this as emergent behavior |

---

## Remaining Divergences

### 1. Context Sample in Initial Prompt (our addition, now optional)

**Our approach** (`_build_context_sample()`): By default, sends evenly-spaced 500-char excerpts from the document in the first message. Can be disabled with `--no-context-sample` to match the paper's opaque-context design.

**Paper's approach**: Root LM only knows CONTEXT exists. It must write code to inspect it.

**Trade-off**: The default gives the LLM a head start (knows format, size, structure immediately) but adds tokens to the initial prompt. Use `--no-context-sample` for the paper's strict "context as environment" paradigm.

**2. Multi-level Recursion (our extension)**

**Paper**: Depth-1 only (root → sub-LM, no deeper).

**Ours**: Configurable `max_depth=3` (supports multi-level recursion).

This is an intentional extension beyond the paper's design.

---

## Emergent Strategy Comparison

Both the paper and our prompts describe the same four-phase strategy:

| Phase | Paper | Our Prompts |
|---|---|---|
| **Inspect** | Examine context structure, size, format | "Study the document sample… understand format" |
| **Search** | Regex/keyword filtering to narrow focus | "Start BROAD: `re.findall(pattern, CONTEXT)` with `re.IGNORECASE`" |
| **Chunk** | Uniform or semantic partitioning | "Extract sections: `CONTEXT[start:end]` (aim for 1000-5000 chars)" |
| **Synthesize** | Aggregate sub-LM results + FINAL() | "Combine and call `FINAL(answer)`" |

Our prompts are more prescriptive (explicit chunk size guidance, specific function examples). The paper observes these strategies as emergent behavior that models discover with minimal prompting.

---

## Future Refinement Candidates

Actionable items identified by this comparison:

- **Default to a cheaper model for recursive sub-calls** — leverage `--recursive-model` param with a smaller default
- **Evaluate prompt prescriptiveness** — test whether our detailed prompts help or constrain the LLM compared to the paper's minimal approach