Commit with Reason: Managing Workspace State in Agentic Scientific Workflows

Warren D. Graham[*†] (wdgraham@lanl.gov); Nathan A. DeBardeleben (ndebard@lanl.gov), PhD[†]
[*]Coastal Carolina University, Department of Computing Sciences; [†]Los Alamos National Lab, HPC-DES

URSA Overview

What is URSA?

URSA is a modular **agentic** ecosystem for automating scientific and engineering tasks. The core idea is to let **small, specialized agents** be combined into a **pipeline** to address tasks from running physics simulations to data analysis. This include separate agents for tasks say as research, planning, code writing and execution, and version control, among others. By coupling **Large Language Model (LLM) reasoning** with domain-specific tools, URSA aims to **accelerate research workflows** and make them more reproducible.
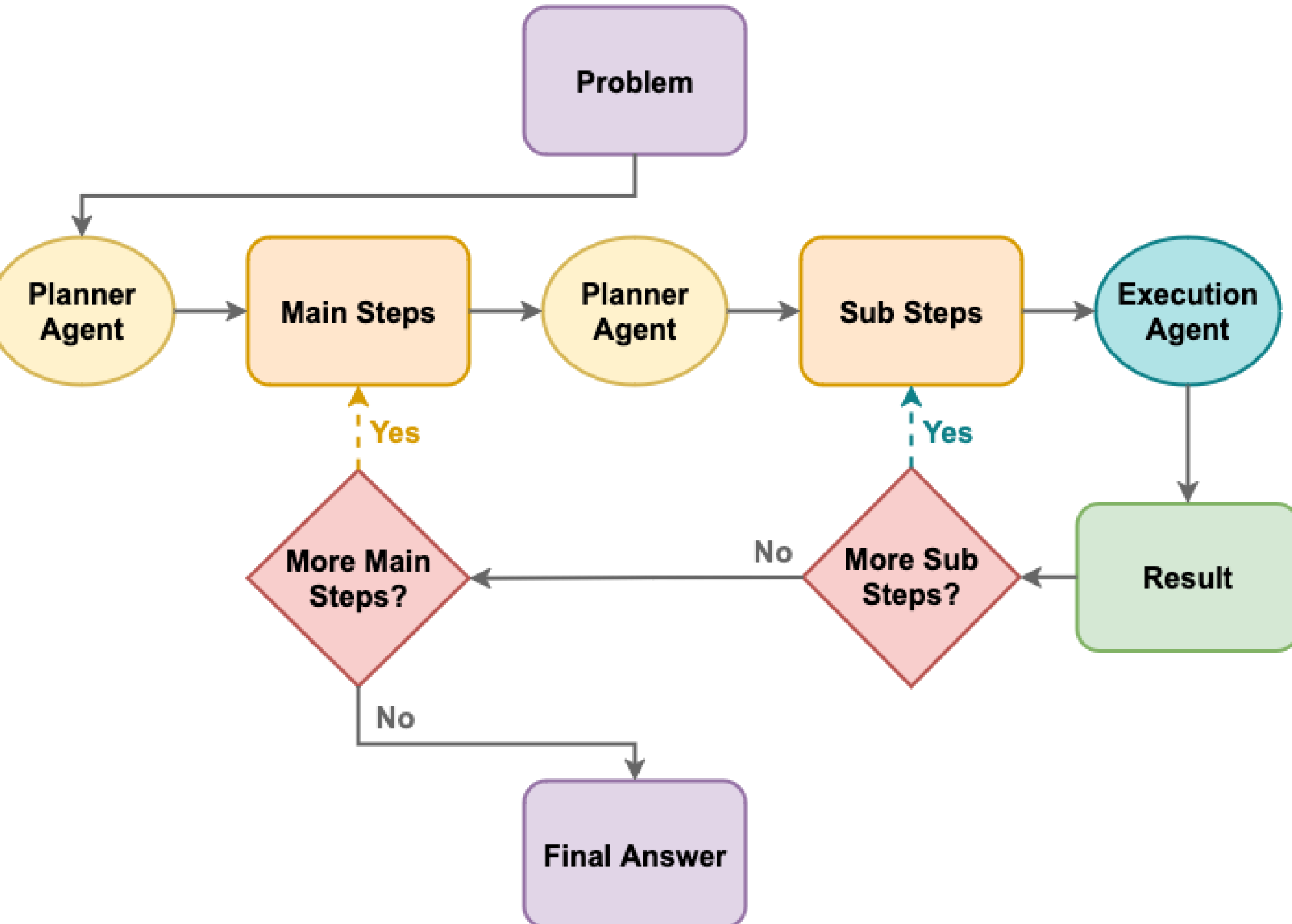

Figure 1a. Example URSA workflow diagram

Execution Agent

Targeted Edits

The original **execution agent** could only overwrite entire files. I added an **edit_code tool** that takes a pair of old and new strings, applies a single replacement, then renders and **prints a unified diff** so the user can see exactly what changed.

Figure 1b. Unified-diff preview

Safety Checks

Previously batched **run_cmd** tool calls in the **execution agent** and the new **git agent** bypassed safety checks. Now every command, including multiple calls in one LLM response, is sent through an LLM-based **safety filter**. Unsafe commands are blocked and reported to the user.

Figure 2a. Safety check blocking git clean from deleting untracked files.

Why a Git Agent?

Our existing agents can perform tasks such as research, planning and modifying our workspace, but without version control these workspace changes live in a **black-box**. We need to capture **why** each modification happened to determine **where mistakes happen** and **reverse them** automatically or through user interaction.

Figure 2b. Git agent diagram

Git Agent Architecture

LLM-Guided Git Management

After the execution agent completes a step, the git agent is given **access to the workspace** and a **summary of the execution agent's actions**. Using this information and its tool, it can explore the workspace and make **git-related changes**.

Figure 2c. Safety check blocking git clean.

Structured Commit Messages

The git agent can **run any git commands**, or other commands deemed safe and required for **version control management**, via its **run_cmd** tool. When it chooses to create a new commit, it follows a rough layout for how a detailed commit message should be worded.

Figure 3a. Git commit message.

Example Problem

Integer Summation

In our integer sum example we ask the agents to **sum the first N integers** using three methods: **a for-loop, Python's built-in sum, and the constant-time formula** and benchmark them. The execution agent codes and benchmarks them while the git agent manages the repo; the shown output confirms **all three give the same result**.

Figure 3b. User running the final python file.

Git Log Visualization

The graph shows **agent-generated commits and merges**; the execution agent summarizes its actions, and the git agent uses that context to reason about **when to commit, merge, or roll back**, writing meaningful messages for later review.

Figure 3c. Git tree after problem is complete.