A Scalable Intelligent Agent System for Automated Monitoring and Debugging Support of the JUNO Electronics system.

Nizar Mahri, Yifan Yang

Université libre de Bruxelles (ULB) — Inter-university Institute for High Energies (IIHE)

25th IEEE Real Time Conference (RT2026)

Introduction & Motivation

The Jiangmen Underground Neutrino Observatory (JUNO) is a 20-kiloton liquid-scintillator neutrino detector designed to determine the neutrino mass ordering. Its central detector is instrumented with 17,596 20-inch PMTs, read out through 5,878 GCUs, 123 BECs, and 8 RMUs. Once sealed and filled, the detector cannot be physically accessed during its projected lifetime, making hardware faults detectable only through statistical analysis of the data stream. This creates a **diagnostic challenge**: physicists must select the right analysis tools, provide correct parameters, interpret outputs and chain results across a heterogeneous tool suite.

This work. Small locally deployed Large Language Models (LLMs) required by this work, exhibit well-documented failure modes when used as tool-calling agents:

- they confuse argument types;
- they select plausible but incorrect tools;
- they lose intermediate results over long tool chains.

We introduce a ***Select-then-Validate*** architecture based on ***Skill Cards:***

- structured descriptions encoding each tool's purpose, parameters, usage constraints, and output schema
- serve as the interface between the LLM planner and the tool suite
- can be automatically synthesized from raw Python functions by an **AI *Learning Pipeline***, adding scalability and removing the need for manual prompt engineering as new tools are added.

The Skill Card Abstraction

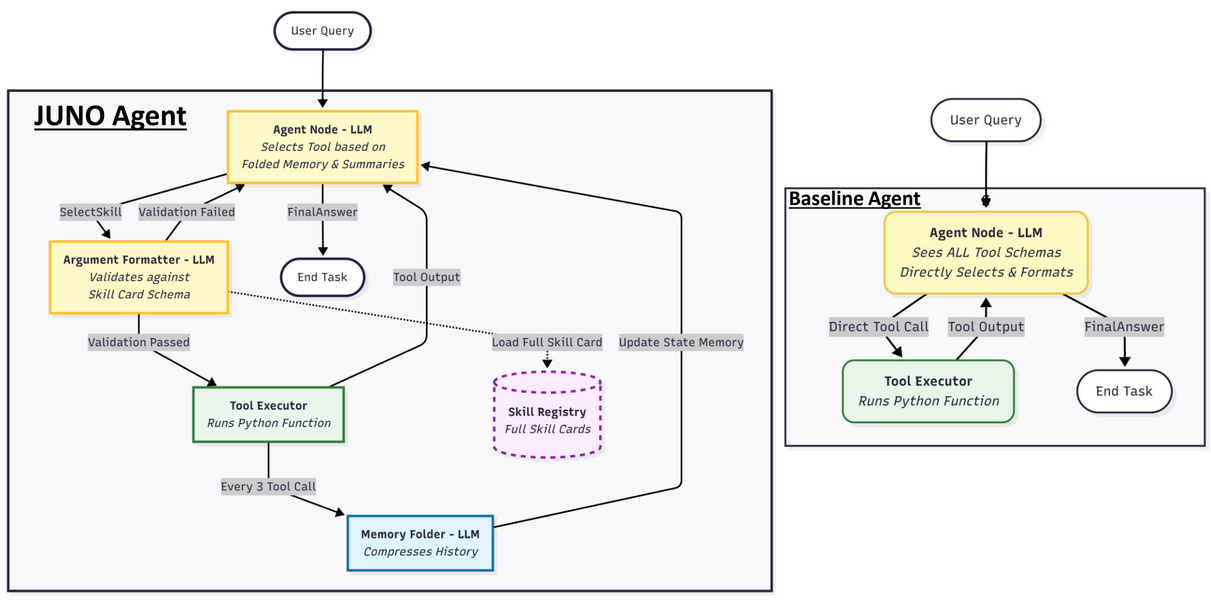
```yaml
skill_id: get_hardware_connections
summary:
  name: get_hardware_connections
  description: >-
    Finds all connected hardware components (PMTs, GCUs,
    BECs, RMUs) for a single JUNO hardware element. Use
    this to trace connections, find related components,
    and localize the scope of a problem.
  parameters:
    type: object
    properties:
      element_type:
        type: string
        description: "The type of hardware element."
        enum: ['PMT', 'GCU', 'BEC', 'RMU']
        required: true
      element_id:
        type: string
        description: "The specific ID of the element."
        required: true
execution_details:
  type: python_function
  call:
    module: "juno_tools.hardware_check"
    function: get_hardware_connections
usage_policy:
  use_when: >-
    - You have a specific component ID and need to find
      what it is connected to.
    - This is the only tool that can trace connections.
  do_not_use_when: >-
    - You are checking for activity in an event window
      (use 'check_component_activity').
output_schema:
  description: "Dictionary of connected hardware."
  type: object
  properties:
    query: { type: "string" }
    connections:
      type: object
      properties:
        pmt_ids: { type: "array", items: { type: "integer" } }
        gcu_ids: { type: "array", items: { type: "integer" } }
        bec_ids: { type: "array", items: { type: "integer" } }
        rmu_ids: { type: "array", items: { type: "string" } }
```

The Skill Card is a *typed, human-readable YAML document* serving simultaneously as:

- tool description for the agent;
- input contract for argument validation;
- execution manifest for dynamic invocation;
- editable documentation for physicists.

Type coercion: Parameters are converted to dynamic Pydantic models at runtime. The LLM's string "4258" is automatically coerced to the integer 4258, addressing **type confusion** without relying on the LLM to produce correct types.

JUNO Agent vs Baseline Agent



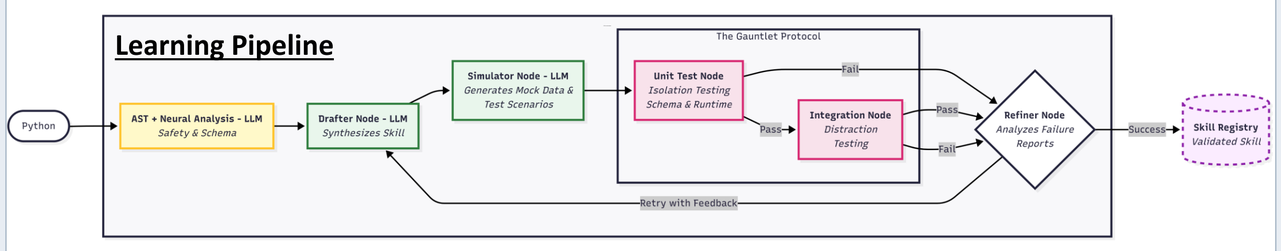
The JUNO agent. Each tool call is split into two LLM passes:

- **Select**: the Agent Node sees the user query, current plan, folded memory, persistent findings, and compact Skill Card summaries. It chooses the next tool and updates the plan.
- **Validate**: the Formatter Node loads the full Skill Card, checks whether the tool is appropriate, and extracts type-coerced arguments before execution.

Folder Memory. A Memory Folder Node periodically compresses the message history into a short summary and an append-only findings list, preserving long-task context without overloading the LLM context window.

Baseline agent. The baseline uses a standard ReAct loop over the same tool suite: one LLM call per iteration, raw tool docstrings, no validation gate, no explicit plan, and no memory compression. Both agents use the same local GPT-OSS 20B model and the same Python tool executor, isolating architecture as the benchmark variable.

Learning Pipeline

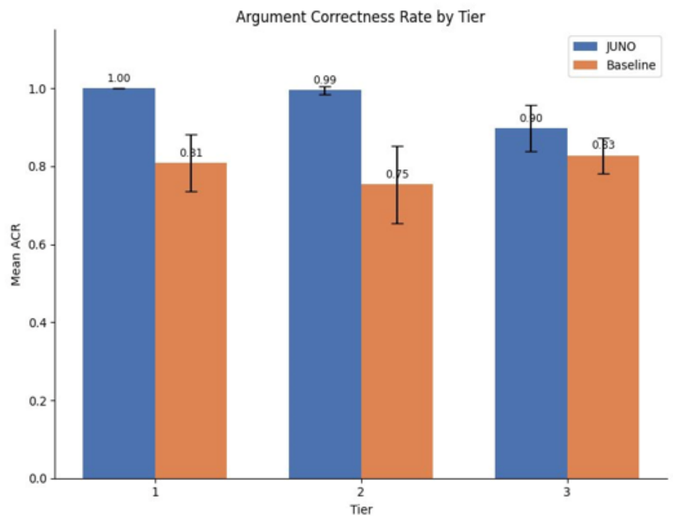


The Learning Pipeline ingests a raw Python function and produces a validated Skill Card.

- *Static Extraction* extracts function metadata via Python AST parsing.
- *Neural Analysis* traces the source code through an LLM.
- *Drafter Node* synthesizes the complete Skill Card from the combined evidence.
- The draft is then validated through unit and integration testing: the pipeline generates test scenarios.

Refinement loop: On failure, Refiner Node generates targeted feedback (distraction analysis, schema fixes) and routes back to Drafter.

Results: Tiered Benchmark



The ACR benchmark comprises diagnostic queries organized into *three tiers of increasing complexity :*

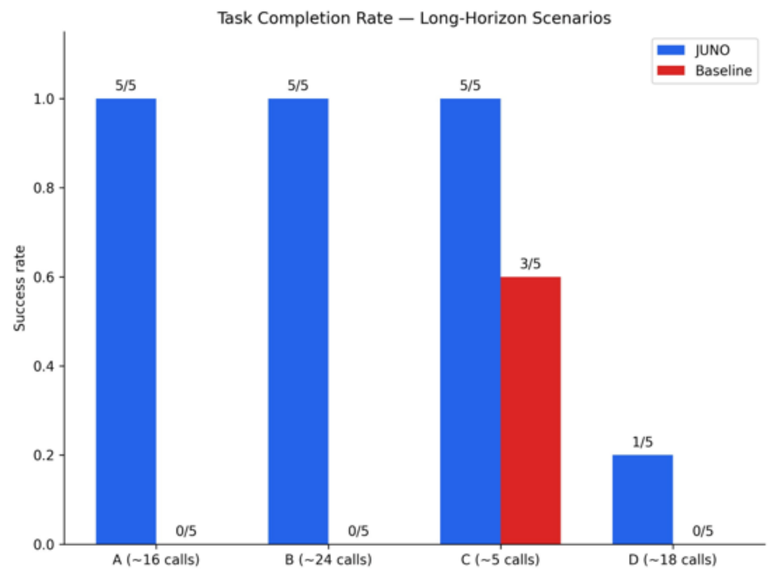
- **Tier 1 -- Direct single-tool calls:** a simple query explicitly maps to one tool.
- **Tier 2 -- Indirect single-tool calls:** the correct tool is still unique, but the agent must infer missing parameters.
- **Tier 3 -- Multi-tool chains:** the query requires sequential tool use.

Argument Correctness Rate (ACR). average fraction of accurately parameterized arguments for tool calling.

Key finding: The JUNO agent's near-perfect ACR confirms that the validation gate and type coercion eliminate argument-level failures.

Long-horizon scenarios. Four realistic diagnostic workflows test whether the agent can preserve context, reuse intermediate results and synthesize a final answer across many tool calls. Each scenario encodes a realistic diagnostic task with a known ground truth.

- A -- persistent flashing PMT
- B -- Temporarily silent RMU
- C -- Filtered local analysis and visualization
- D -- Comprehensive detector health check



JUNO vs Baseline. The baseline cannot solve tasks beyond ~5 sequential tool calls. Failure modes: synthesis gap, structural boundary error and tool confusion under catalogue pressure. Scenario D (long and demanding cross-subsystem check) defines the current capability ceiling.

Conclusions & Future Work

The Select-then-Validate architecture, paired with Skill Cards and Folded Memory, achieves near-perfect argument correctness and enables reliable long-horizon diagnostics on small locally deployed models.

The Learning Pipeline demonstrates that Skill Cards can be automatically synthesized from raw Python functions via AST analysis, LLM-based code tracing, and iterative validation with targeted feedback.

Future directions includes stronger local models, adaptive memory compression, deployment on live monitoring streams, and testing whether Skill Cards transfer to other large detector experiments.

nizar.mahri@ulb.be yang.yifan@ulb.be | Models: GPT-OSS 20B (local, Ollama) | Stack: LangGraph · Python · Pydantic · JUNOSW