Integrating AI into My Existing ETL Pipeline System

I recently experimented with integrating an AI agent into an ETL system I built previously. The goal was to see whether an agent could simplify the workflow logic while still leveraging the existing data processing tools. The result was a much shorter orchestration layer and a surprisingly clean architecture.

The original pipeline was functional and reliable, but like many ETL systems, it required explicit workflow logic to control how each stage of processing moved from one step to the next.

My goal was to explore whether an **AI-driven orchestration layer** could simplify the design while still leveraging the existing ETL tools I had already developed.

The result was surprisingly effective. By introducing an agent to manage workflow decisions, the pipeline architecture became significantly simpler. Instead of hard-coding the order of execution

for each step, the agent evaluates the **current state of the data** and decides which tool should run next.

Once the first tool was configured and working within the agent framework, the rest of the tools followed the same pattern and were relatively straightforward to integrate. The system I built includes several core components. At the center is a **state-driven agent loop**, which continuously evaluates the state of the pipeline until processing is complete. A **tool registry** defines the available operations that the agent can execute, and a routing function determines which tool should run based on the current state of the data.

The tools themselves were adapted from components I developed during 2025 as part of a healthcare ETL pipeline system. These tools perform the actual data processing tasks, including:

- Parsing HL7 messages
- Extracting segments from HL7 messages
- Transforming segments into FHIR resources
- Building a FHIR bundle
- Persisting the resulting data into a MySQL database

In addition to integrating these tools into the agent framework, I implemented **routing logic through a function that determines which tool should execute next.** Debugging the routing logic and managing state transitions became an important part of ensuring that the pipeline executed cleanly from start to finish.

With this structure in place, the AI agent now handles the orchestration of the workflow. The agent routes the pipeline dynamically through the stages of ingestion, parsing, extraction, transformation, and persistence of patient data derived from HL7 messages.

A typical execution of the pipeline now looks like this:

PIPELINE EXECUTION
1 ingest_file
2 parse_hl7_tool
3 extract_patient_tool
4 transform_patient_fhir_tool
5 persist_bundle_tool
6 save_result

PIPELINE COMPLETE

Program code that shows the main control loop for the agent

```
def run_agent(input_data):
    max_steps = 20
    steps = 0
    state = {
        "file_path": input_data,
        "raw_data": None,
        "fhir_patient": None,
        "hl7": None,
        "patient": None,
        "fhir_resources": [],
        "persisted": False,
        "complete": False,
        "history": []
    }
    while not state["complete"] and steps < max_steps::
        tool_name = ask_agent(state)
        print("Agent chose:", tool_name)
        print("State before tool:", state)
        tool_name = ask_agent(state)
        tool = TOOLS[tool_name]["function"]
        state = tool(state)
        state["history"].append(tool_name)

        print("State after tool:", state)
        steps += 1
    print("\nPIPELINE EXECUTION")
```

Each step represents a different state of the data as it moves through the pipeline. The agent evaluates the state after each step and determines which tool should run next until processing is complete.

The data transition from raw data to finished FHIR resource

Raw HL7 Data

MSH|^~\\&|HIS|GENERALHOSPITAL|EHR|GENERALHOSPITAL|202603151230||ADT^A01|MSG00001|P|2.5\nEVN|A01|202603151230\nPID|1||123456^^^GENERALHOSPITAL^MR||DOE^JOHN^A||19800101|M|||123 MAIN ST^^VERNON^BC^V1T1A1||555-123-4567|||S||123456789|987654321\nNK1|1|DOE^JANE|SPOUSE|123 MAIN ST^^VERNON^BC^V1T1A1|555-987-6543\nPV1|1|I|W^389^1^GENERALHOSPITAL||||1234^SMITH^ROBERT^J|||MED|||||||1234567|SELF||||||||||||||||||||||||202603151200\nAL1|1||^PENICILLIN||PRODUCES HIVES\nDG1|1||J18.9^PNEUMONIA^ICD10|||A'}

FINAL STATE OUTPUT

FHIR PATIENT RESOURCE:


```
{
  "resourceType": "Patient",
  "identifier": [  {
      "system": "http://hospital.example.org/mrn",
      "value": "123456^^^GENERALHOSPITAL^MR"
    }  ],
  "name": [   {
      "family": "DOE",
      "given": [
        "JOHN"
      ]  }  ],
  "gender": "male",
  "birthDate": "1980-01-01" }
```


One of the most interesting aspects of this approach is how it separates **workflow orchestration from data processing logic**. The ETL tools remain focused on performing specific tasks, while the AI agent manages the flow of execution. This results in a much simpler orchestration layer and provides a flexible foundation for extending the pipeline in the future.

For example, additional HL7 segments such as diagnoses, allergies, or encounters can be incorporated by simply adding new extraction and transformation tools to the registry. The agent can then route the workflow through those tools as needed based on the state of the data.

I added an interactive layer that allows the user to specify input data and generate analytics reports after pipeline execution.

This experiment demonstrates a promising pattern for building **next-generation data integration pipelines**, particularly in healthcare environments where standards such as HL7 and FHIR are widely used. By combining traditional ETL components with an agent-driven orchestration layer, it becomes possible to build pipelines that are both simpler and more adaptable. I'm continuing to explore how agent-based architectures can **simplify data integration pipelines.**