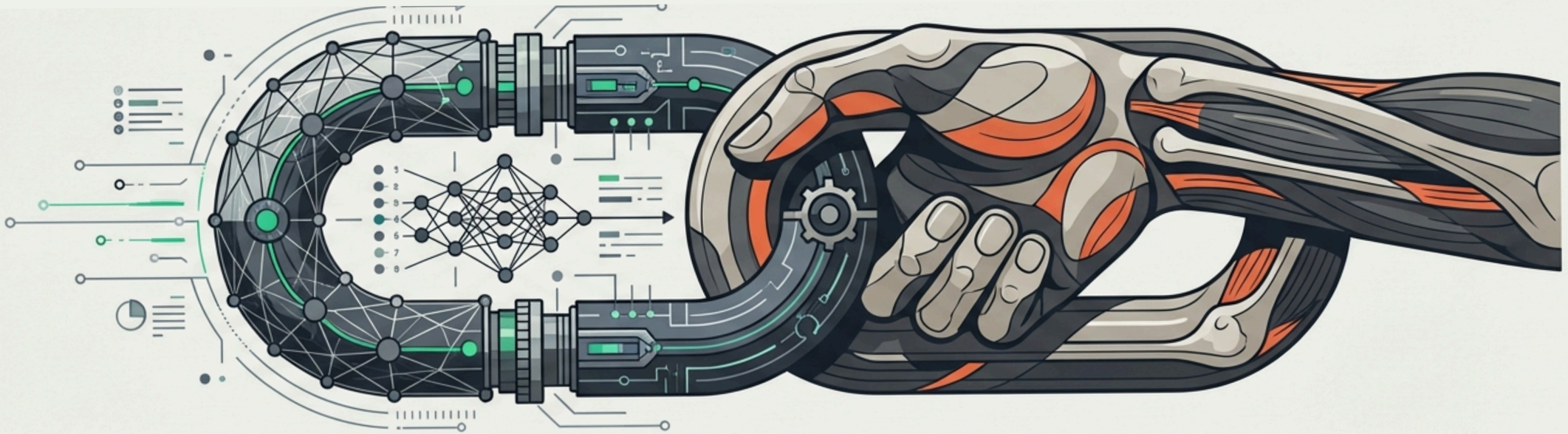
AuthChain

By WillOWisp at HackTU 7.0

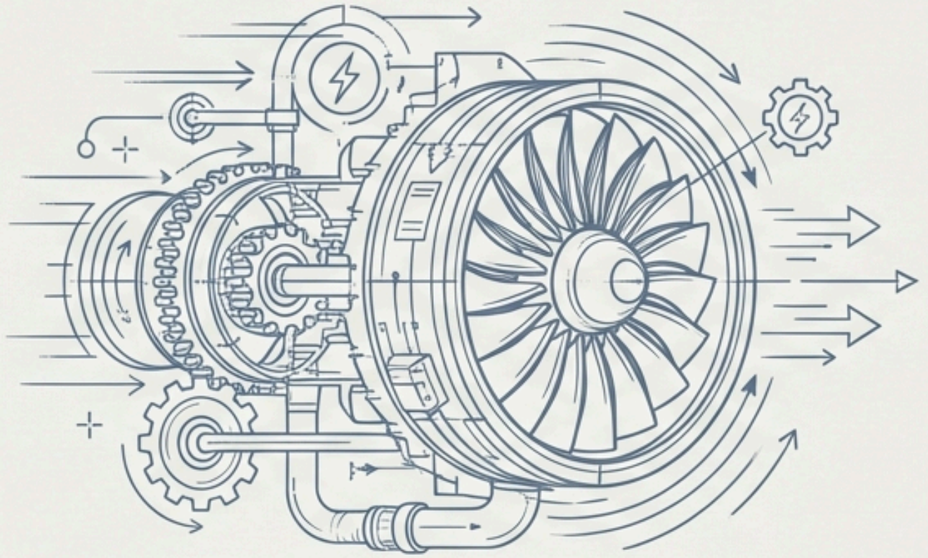
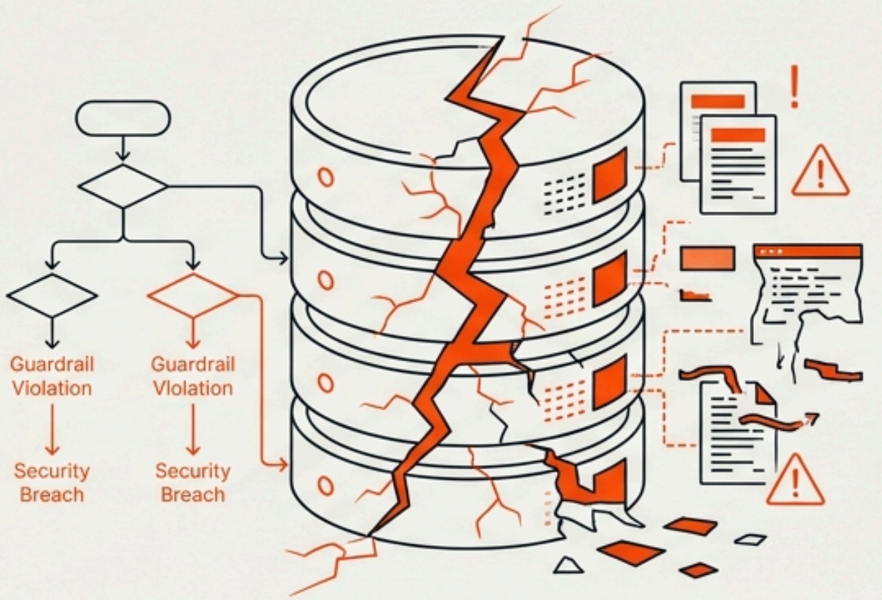


Immutable Governance for Autonomous AI Systems

Bridging the gap between AI utility and human oversight via blockchain-validated workflows

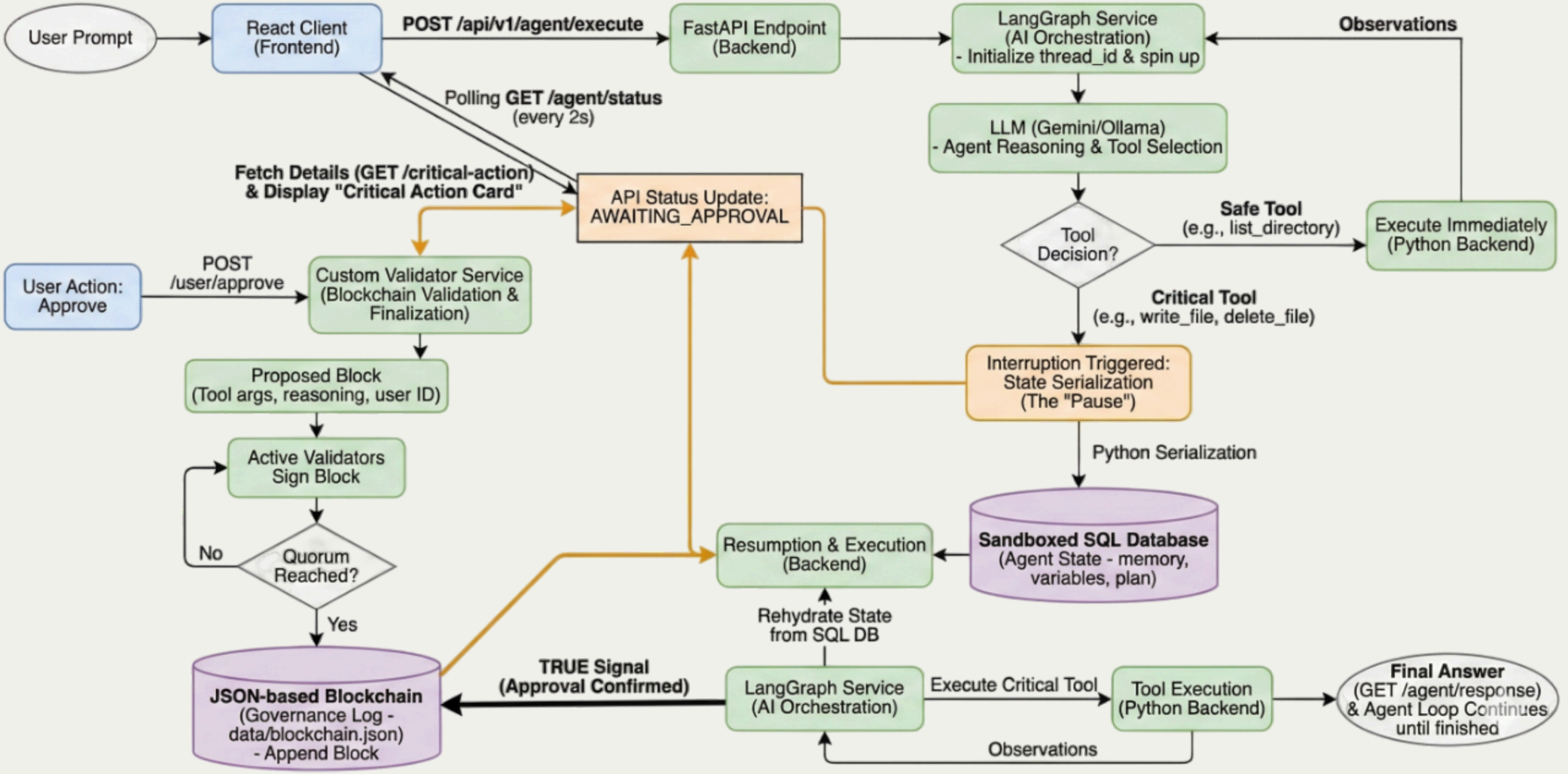
Contributors: Arnab Mandal, Suryansh Rohil, Shresth Tiwari

The Conflict: Autonomy vs. Control

High-Frequency Utility	Probabilistic Failure
	
AI excels at rapidly conducting low-complexity tasks. It provides speed and automation scale.	LLMs are probabilistic, not deterministic. This allows them to bypass prompt guardrails and execute tasks that violate security and privacy of data.
We need the automation of the agent, but the immutability of a blockchain log for sensitive actions.	

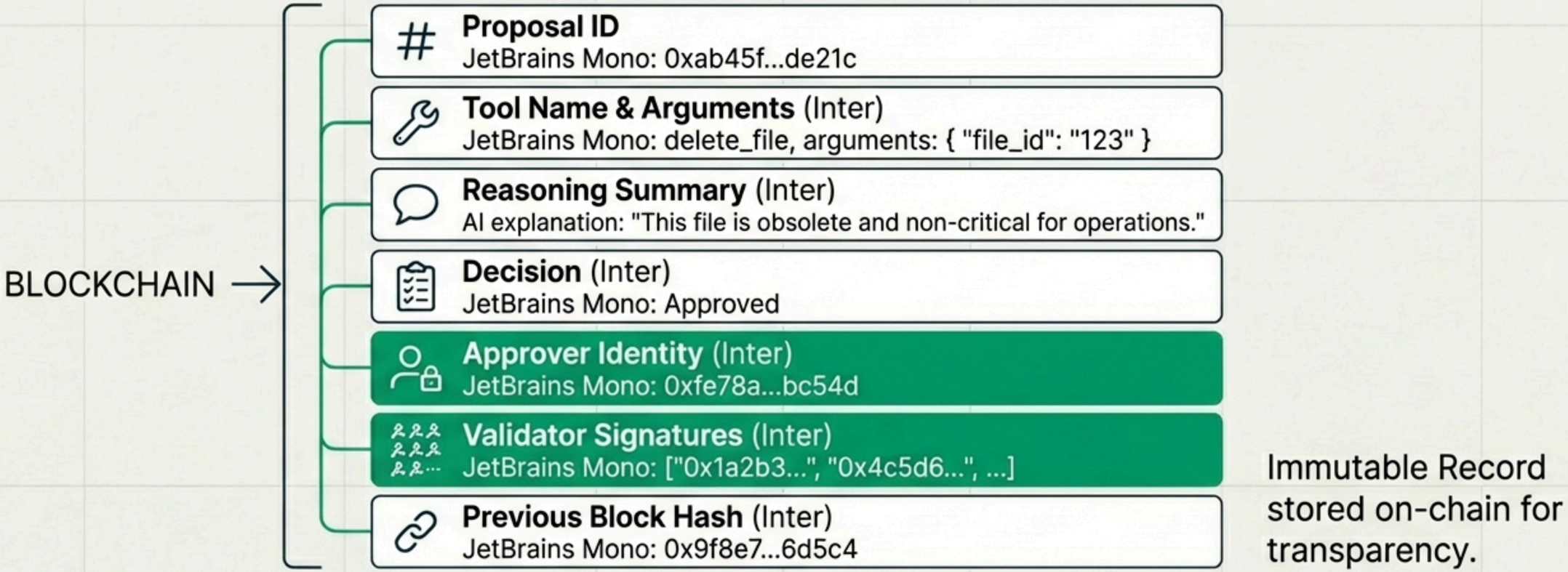
Interruption-Driven Control Architecture

AuthChain is not just a log; it is a gatekeeper.



Anatomy of a Governance Proposal

The immutable record of ‘Who’, ‘What’, and ‘Why’.



Workflow A: The Safe Execution Path Execution Dlan

User Request

API: RUNNING
JetBrains Mono

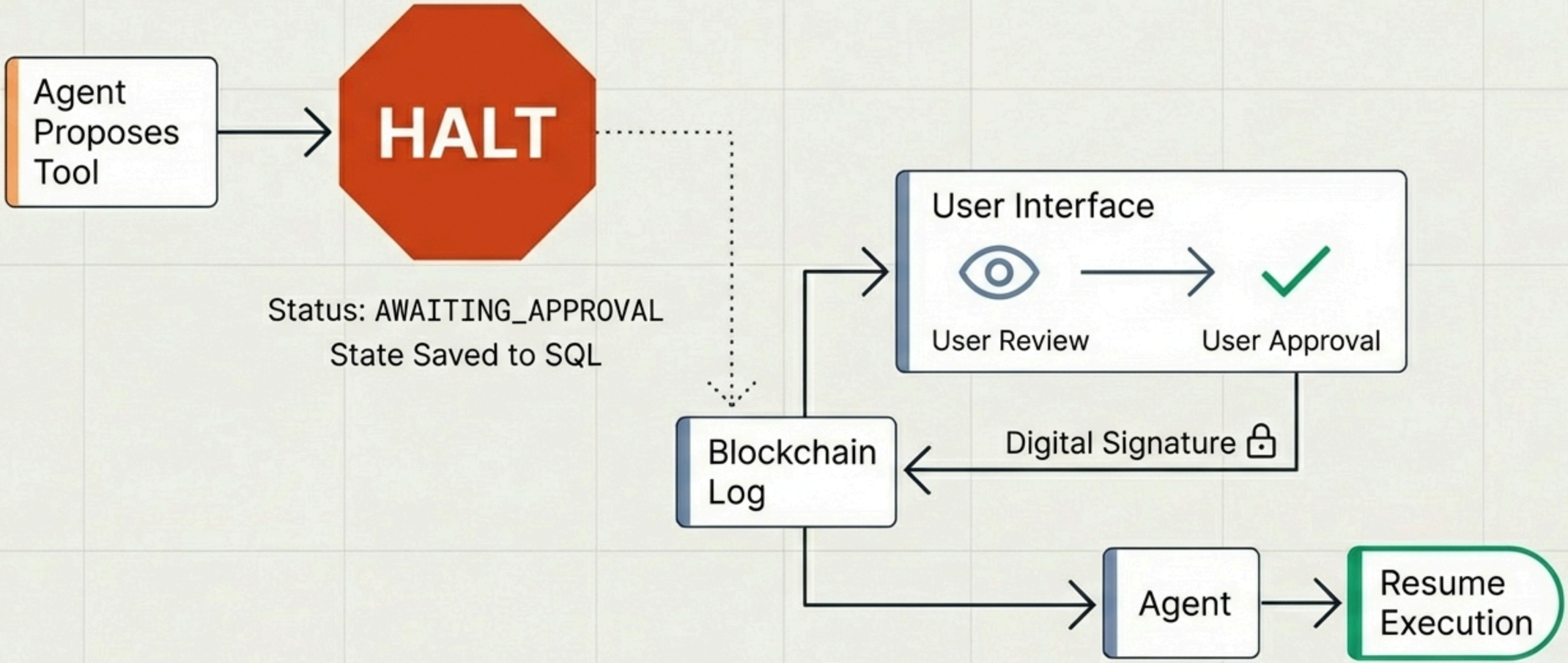
Agent
→ Safe Tool
(Read)

Execution

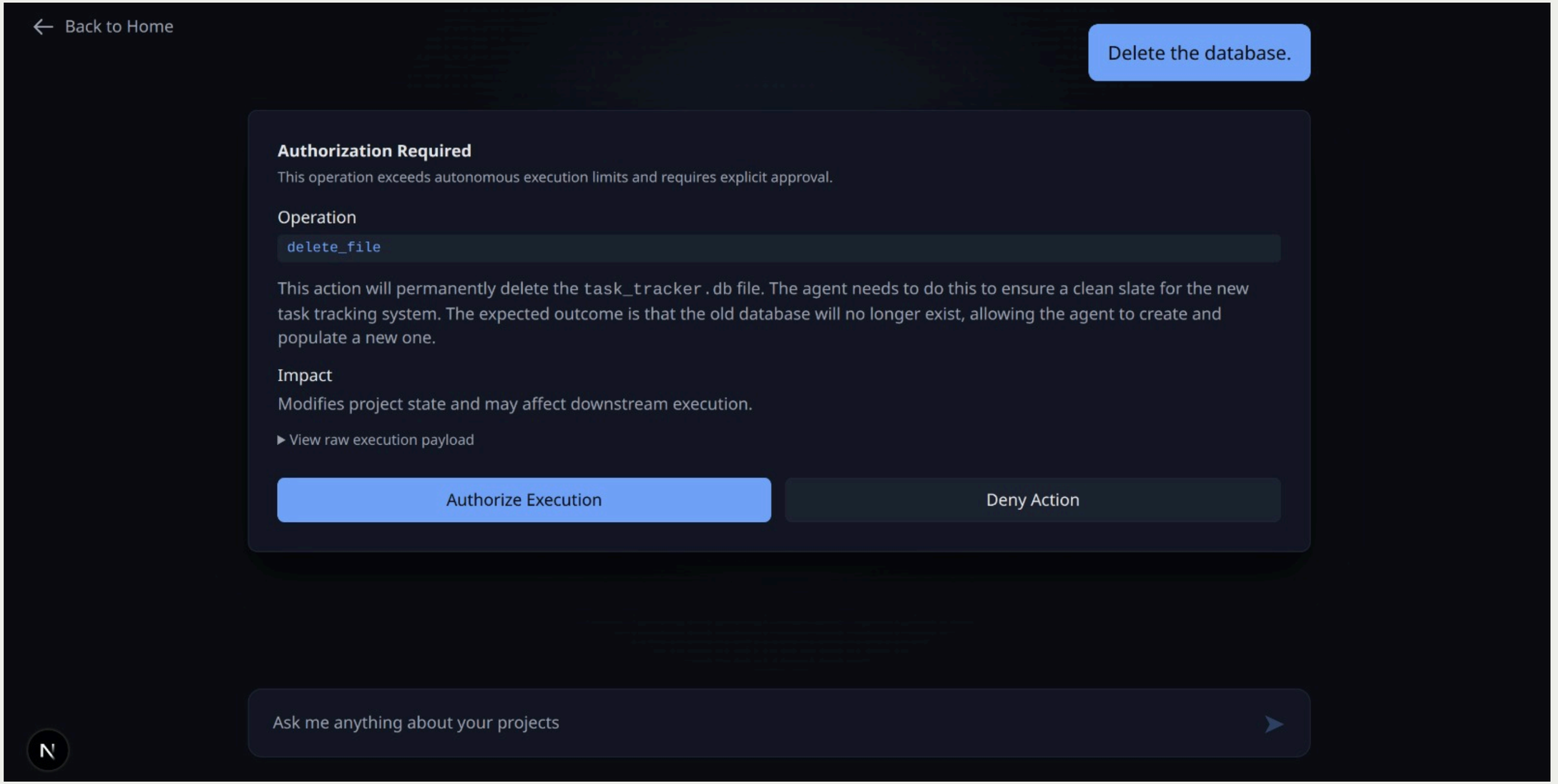
Response

Speed is maintained for non-destructive actions. No interruption occurs.

Workflow B: The Critical Execution Path



The Human-in-the-Loop Interface



LEVERAGING POLICY SERVICES FOR DYNAMIC TOOL CREATION

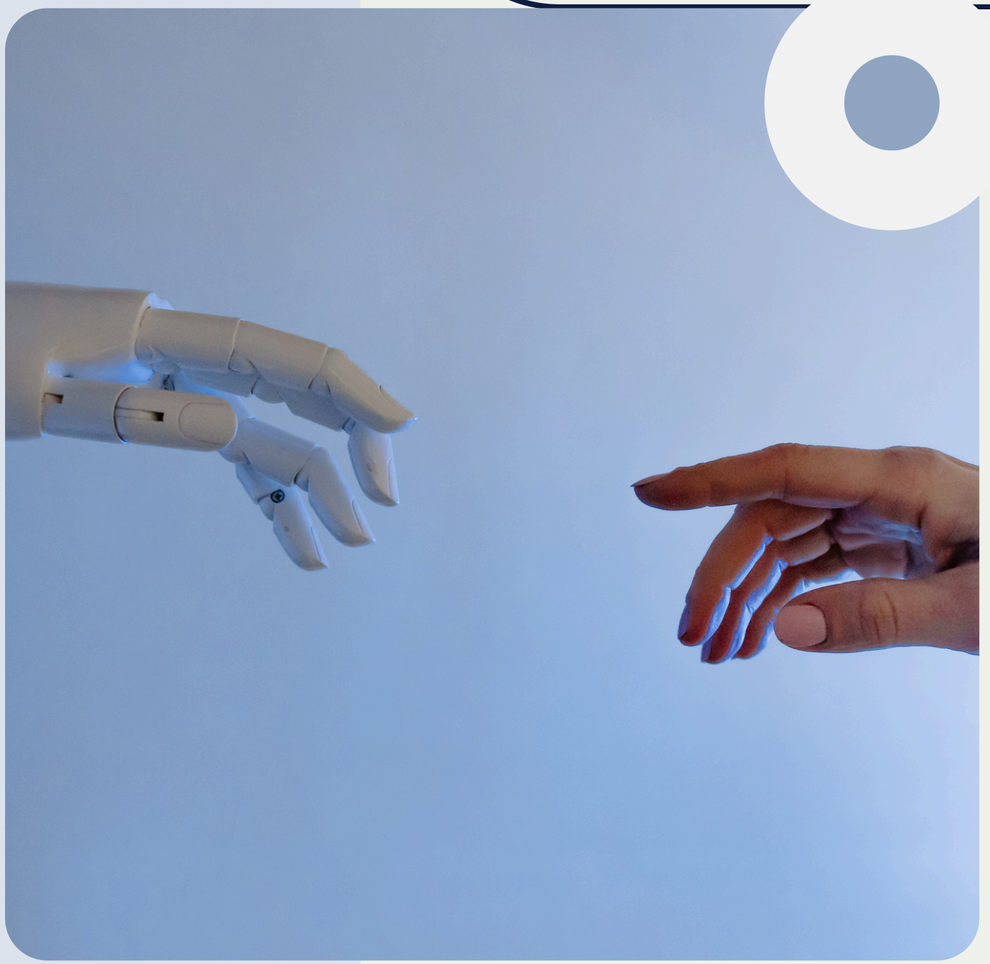
This is a very rudimentary prototyped feature

```
[2/4] Validating tool schema and structure...
Safe

[3/4] Generated code:
--------------------------------------------------------------------------------
from langchain_core.tools import tool

@tool
def compute_rsi_score(prices: list, period: int) -> str:
    """
    Calculates Relative Strength Index (RSI) score based on closing prices

    Args:
        prices: List of closing prices
        period: Time period for RSI calculation

    Returns:
        Result message as string
    """
    try:
        gains = [p - p_prev for p, p_prev in zip(prices[1:], prices[:-1]) if p > p_prev]
        losses = [-p + p_prev for p, p_prev in zip(prices[1:], prices[:-1]) if p < p_prev]
        avg_gain = sum(gains) / len(gains) if gains else 0
        avg_loss = sum(losses) / len(losses) if losses else 0
        rs = avg_gain / avg_loss if avg_loss != 0 else 0
        rsi = 100 - (100 / (1 + rs))
        return f'RSI: {rsi:.2f} ({period}-period)' if len(prices) >= period else 'Not enough
    except Exception as e:
        return f'ERROR: {str(e)}'

--------------------------------------------------------------------------------

[4/4] Registering...
      ✓ Saved with hash: cbbde74992711741

Tool 'compute_rsi_score' is ready to use!
```

```
Safe

[3/4] Generated code:
--------------------------------------------------------------------------------
from langchain_core.tools import tool

@tool
def build_rust_to_python_server_api(api_name: str, port_number: int) -> str:
    """
    Generates a basic Rust-to-Python API server structure

    Args:
        api_name: Name of the API
        port_number: Port number for the server

    Returns:
        Result message as string
    """
    try:
        import flask
        app = flask.Flask(__name__)
        @app.route('/', methods=['GET'])
        def home():
            return f'Welcome to {api_name} API'
        app.run(port=port_number, debug=True)
        return 'API server started successfully'
    except Exception as e:
        return f'ERROR: {str(e)}'

--------------------------------------------------------------------------------

[4/4] Registering...
      ✓ Saved with hash: 6712df75cee2bc47

Tool 'build_rust_to_python_server_api' is ready to use!
```

```
[2/4] Validating tool schema and structure...
Safe

[3/4] Generated code:
--------------------------------------------------------------------------------
from langchain_core.tools import tool

@tool
def build_sql_view_tool(table_name: str, view_columns: list) -> str:
    """
    Generates SQL code to create a database view from a given table and columns

    Args:
        table_name: Name of the source table
        view_columns: List of column names to include in the view

    Returns:
        Result message as string
    """
    try:
        sql_code = f'CREATE VIEW {table_name}_view AS SELECT '

        for col in view_columns:
            sql_code += f'{col}, ' if col != view_columns[-1] else f'{col};'

        return sql_code
    except Exception as e:
        return f'ERROR: {str(e)}'

--------------------------------------------------------------------------------

[4/4] Registering...
      ✓ Saved with hash: abb25c9ccb1c206d

Tool 'build_sql_view_tool' is ready to use!
```

AUTHCHAIN: GOVERNANCE FOR AUTONOMOUS AI



1. Governance & Control

Regulates AI agent permissions and enforces *human-in-the-loop approvals* for sensitive production actions.

2. Enterprise Security & Compliance

Provides *full audit trails* and automated policy enforcement for highly regulated environments.

3. High-Value Integration

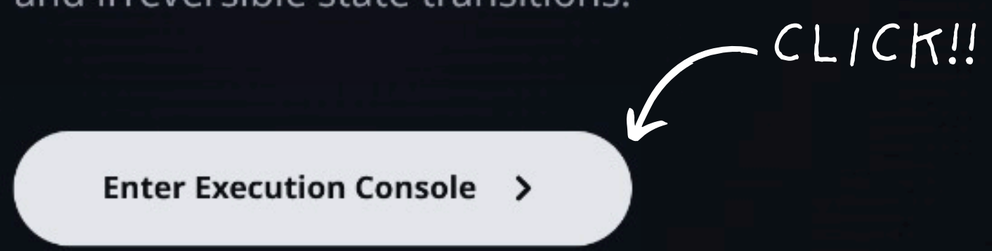
Embedded security for GitHub, GitLab, Cursor, and CI/CD, bringing safety to real-world developer workflows.

By WillOWisp, at HackTU 7.0

Github

AuthChain

Deterministic agent execution enforced by **on-chain governance**
and irreversible state transitions.



Because True AI autonomy requires immutable governance

Contributors: Arnab Mandal, Suryansh Rohil, Shresth Tiwari