Airflow as an AI Agent's toolkit

Going Beyond MCPs & Unlocking 1000+ Integrations

Kaxil Naik & Pavan Kumar Gopidesu

Introduction

Kaxil Naik
Sr Eng Director @ Astronomer
Airflow Committer & PMC Member

Pavan Kumar Gopidesu
Technical Lead @ Tavant
Airflow Committer & ASF Member

24+ hours of work!

Every. Schema. Change.

2-3 times per week

Customer use case - Relies on a lot of incoming data

Data Sources

- Hundreds of clients → thousands of S3, GCS feeds
- Formats vary (Parquet/CSV/JSON); schemas evolve
- Frequent schema drift
- Data across clouds

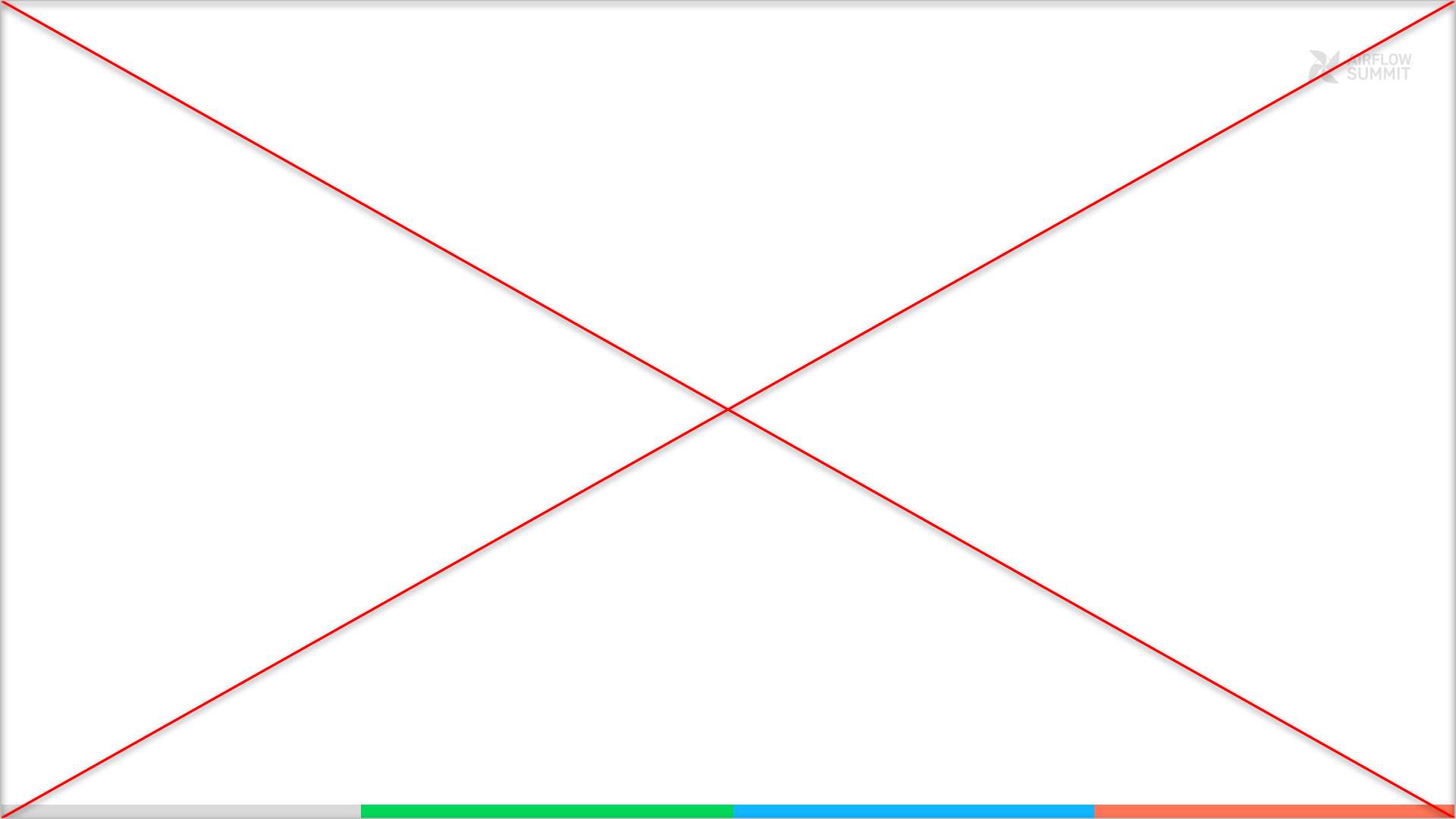
Consumers:

- Per-consumer tables (Postgres, Iceberg, Glue)
- Upstream drift breaks ingestion

When things break!

- Debug!
- Schema comparison across files & DB
- Change order processes
- Manual fixes
- Backfills
- Notifying 100+ consumers

Demo: Handling Schema Drift



Demo: Data Quality Checks

Airflow | Queues | Simple Queue Serv

localhost:28080/dags/llm_dq_check_s3_feed

Incognito

Dag
llm_dq_check_s3_feed

Search Dags ⌘+K | Trigger

Home
Dags
Assets
Browse
Admin
Docs
User

Options

78s
39s
0s

CustomerDataS3Feed
ApproveCustomerDataDQResults

llm_dq_check_s3_feed

Favorite Dag

Schedule	Latest Run	Next Run	Owner	Tags	Latest Dag Version
	2025-10-06, 12:47:15		airflow		v1

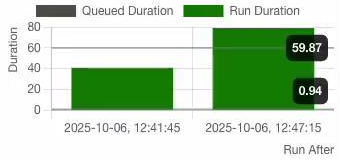
Overview | Runs | Tasks | Calendar | Required Actions | Audit Log | Code | Details

Last 24 Hours 2025-10-05, 13:24:37 - 2025-10-06, 13:24:37

0 Failed Tasks

0 Failed Runs

Last 2 Dag Runs



What currently takes 24+ hours:

❌ Manual schema comparison

❌ Change order approval

❌ Code changes and testing

❌ Debugging and backfills

What this PROTOTYPE demonstrates:

✅ Automated detection (using Apache DataFusion & AI)

✅ Human oversight at critical points

✅ Cross-cloud validation

✅ Business-friendly explanations

Does this problem resonate?

Would this solution work for you?

How does this work?

Automatic Context Injection

For **SQL operations**:

- Database type and version (PostgreSQL 15.2)
- Full schema from `DbApiHook` or `Asset` metadata
- Sample data (first few rows)
- Built-in safety rules

For **File operations**:

- File format (Parquet, JSON, CSV)
- Storage type (S3, GCS, Azure)
- File size, row count estimates
- Schema information
- Partitioning structure

Safety Mechanisms

We are not just sending prompts to LLMs.

Safety layers we're exploring:

- ✅ SQL Safety: Blocks DROP, DELETE without WHERE, TRUNCATE
- ✅ Human-in-the-Loop: Required for sensitive operations
- 🚧 Query validation: Parse and analyze before execution
- 🚧 Asset sensitivity: Mark Assets as auto-requiring approval for accessing it (PII)
- 🚧 Audit logging: All AI decisions tracked separately
- 🚧 Read-only by default: Write operations need explicit approval

Why Apache DataFusion?

- Unified query engine across object stores and DB. (S3, Postgres)
- Multiple formats (Parquet, JSON, CSV, Iceberg, Delta Lake)
- Single-node performance (no Spark overhead)
- Performance (in our test): 50M records in 14 seconds (with joins, groupby, min, max etc)

⚠ DataFusion is for READING only. Write uses `DBApiHook`

Current Approach - Specialized Operators

Current Implementation:

- `LLMSchemaCompareOperator` - for schema drift
- `LLMDataQualityOperator` - for validation
- `LLMFileAnalysisOperator` - for file analysis
- … more to come for interacting with API(s) apart from Files & DB

Why specialized: Clear intent, better context for LLM, type safety, focused documentation

Alternative being explored: Unified `LLMOperator` with resource adapters

We're still figuring out the right abstraction. Your feedback will help.

Integration with Assets

Mark Asset as sensitive

Define how to access the Asset

- URI
- Connection

Define Asset type

- Data format
- Schema

Define metadata (for better AI context)

- Description
- Example queries

Future:

- Validations
- Statistics

```python
from airflow.sdk import Asset

customer_asset = Asset(
    name="customer_data",
    uri="s3://bucket/customers/",
    conn_id="aws_default",
    schema={
        "customer_id": "int64",
        "email": "string",
        "phone_verified": "boolean"
    },
    sensitivity="pii",
    format_="parquet",
    statistics={"estimated_rows": 50000000}
)
```

Airflow PMC perspective

What Airflow Principles Must Stay

Whatever we build must preserve Airflow's core strengths:

✅ Deterministic DAG structure - static, reviewable, testable

✅ Observable - lineage, logging, monitoring

✅ Reliable - existing retry logic, error handling

✅ Safe - no breaking changes to existing workflows

Leveraging an LLM is just one task in a predictable pipeline.

We're NOT building AI that changes DAG structure.

What We're NOT Building

❌ **NOT**:

AI that changes DAG structure

Dynamic pipeline generation

AI that makes architecture decisions

Replacement for your data engineers

✅ **YES:**

AI for repetitive, context-dependent tasks

Deterministic DAGs with intelligent tasks

Human oversight at critical points

Audit trails and observability

Implementation Reality Check

If we proceed, the path would be:

- Phase 1: Experimental provider (`apache-airflow-providers-ai`)
- Phase 2: Community feedback and iteration
- Phase 3: Production-ready provider (if it proves valuable)
- Phase 4: Core integration (only if community demands it)

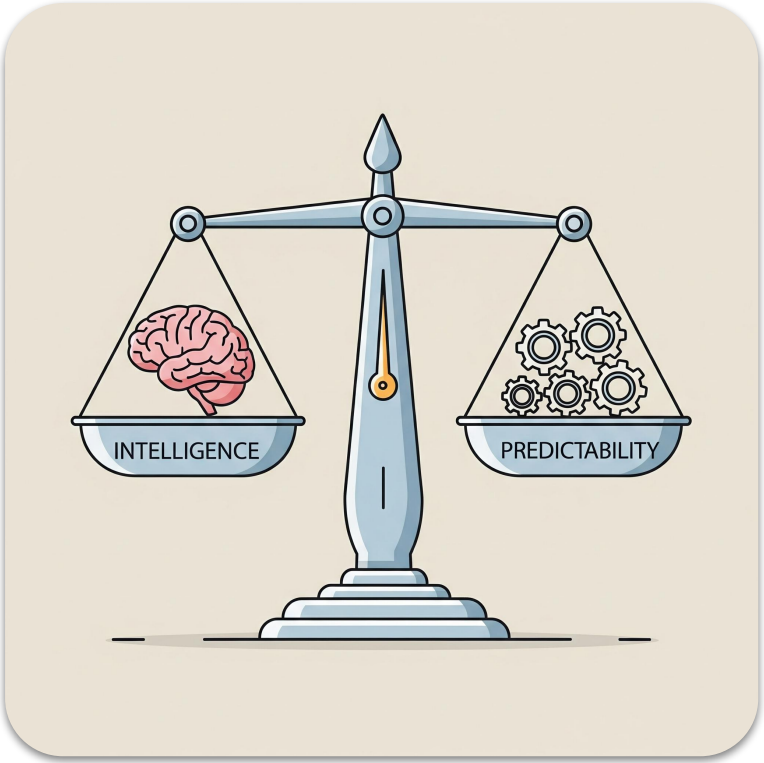
This could take multiple months to get right. No shortcuts.

Should Airflow have production-ready AI operators?

What We Need From You

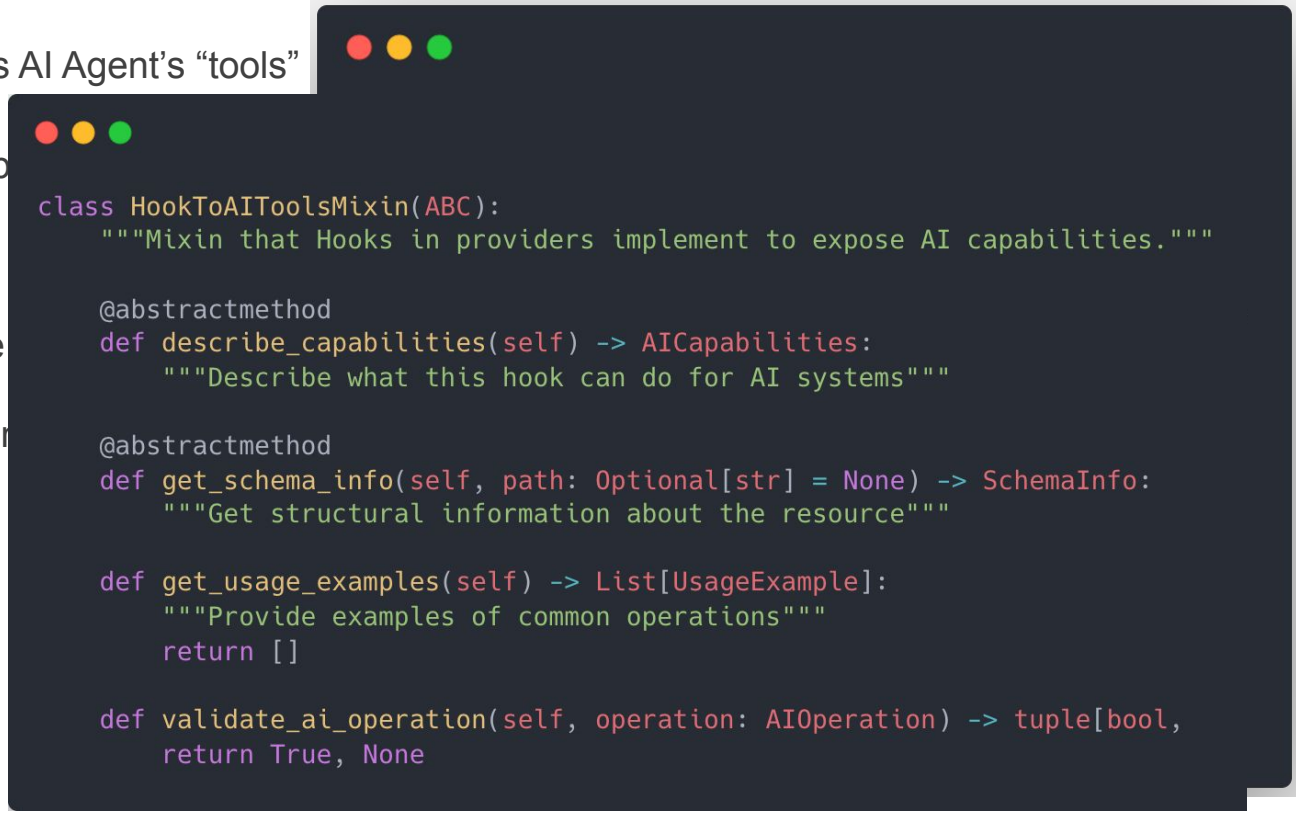
Before we go further, we need community input:

- Is this solving real problems you face?
- What safety mechanisms are non-negotiable?
- How should we handle AI errors and edge cases?
- Right balance between intelligence & predictability?
- Should this be a provider or core feature?



Future Possibilities

- Expose 1000s of Hooks as AI Agent's "tools"
- Could AI-detected issues b
- Should operators propose
- Multi-agent validation (one
- All AI calls can be logged i
- … any other wild ideas (?)



```python
class HookToAIToolsMixin(ABC):
    """Mixin that Hooks in providers implement to expose AI capabilities."""

    @abstractmethod
    def describe_capabilities(self) -> AICapabilities:
        """Describe what this hook can do for AI systems"""

    @abstractmethod
    def get_schema_info(self, path: Optional[str] = None) -> SchemaInfo:
        """Get structural information about the resource"""

    def get_usage_examples(self) -> List[UsageExample]:
        """Provide examples of common operations"""
        return []

    def validate_ai_operation(self, operation: AIOperation) -> tuple[bool,
        return True, None
```

How to get involved?

💬 **Mailing list**: dev@airflow.apache.org (AIP coming after Summit)

🗣️ **Slack**: #airflow-3-dev channel

📧 **Pavan**: gopidesupavan@gmail.com
[He wants your feedback directly]

📧 **Kaxil**: kaxil@astronomer.io

Questions?
Concerns?
Ideas?

AIRFLOW SUMMIT

The 2025 Apache Airflow® Survey is here!

Fill it out to for a free Airflow 3 Fundamentals or DAG Authoring in Airflow 3 certification code