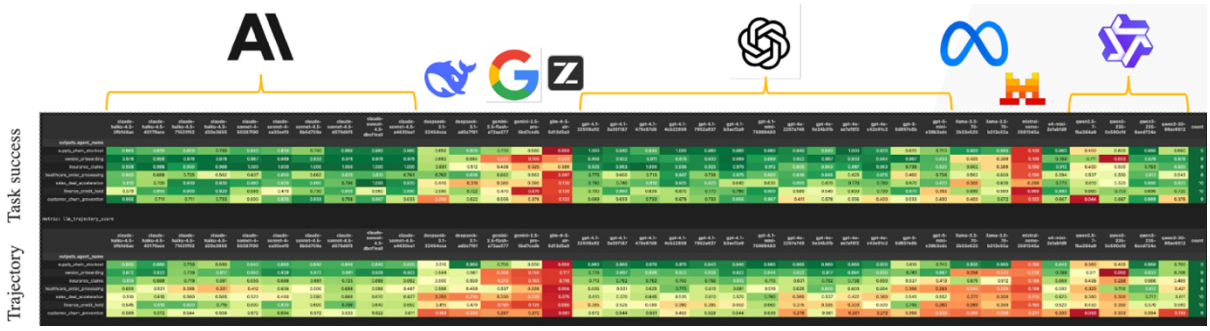
AUTOMATION ANYWHERE

Technical White Paper:
A Framework for Evaluating Goal-Based AI Agents



As enterprises increasingly deploy AI agents for business process automation, the lack of domain-specific evaluation frameworks poses risks to reliability, efficiency, and compliance. This white paper presents a comprehensive evaluation methodology developed through extensive internal benchmarking and real-world deployment experience.

Key Contributions:

- Dual-metric framework measuring both task success and reasoning trajectory.
- Purpose-built dataset architecture for business process automation.
- Creation of our own APA-Eval V5 business process dataset.
- Analysis of common failure patterns, specifically the "Scrappy Win."

1. Introduction

1.1 The Agent Evaluation Gap

Traditional AI benchmarks evaluate linguistic competence: question answering, text generation, and reasoning over static contexts. However, autonomous agents in business environments exhibit fundamentally different characteristics:

- **Multi-step execution:** Agents decompose goals into action sequences.
- **Tool orchestration:** Integration with APIs, databases, and external systems.
- **Stateful reasoning:** Maintaining context across extended workflows.
- **Structured I/O:** Processing and generating business-specific data formats.

Public benchmarks, while valuable for model comparison, fail to capture these dimensions critical to production agent systems.

1.2 Business Process Automation Requirements

Enterprise automation introduces additional constraints:

- **Auditability:** Understanding the reasoning path behind each decision.
- **Efficiency:** Minimizing API costs, latency, and resource utilization.
- **Safety:** Preventing unauthorized actions and cascading failures.
- **Reliability:** Consistent performance across input variations and edge cases.
- **Explainability:** Providing clear justification for agent actions.

1.3 Our Approach

We developed a comprehensive evaluation framework grounded in actual business automation use cases, validated through extensive deployment across multiple domains including customer service, document processing, and workflow automation.

2. Dataset Architecture

To ensure our benchmarks reflect real-world complexity rather than simple Q&A, we developed a four-stage generation pipeline. This pipeline transforms unstructured source material into executable, code-based test suites.

2.1 Stage 1: Source Ingestion

The process begins with raw **Source Documents**. Unlike public benchmarks that often rely on synthetic prompts, our pipeline ingests actual business artifacts: process documentation, API references, code snippets, and unstructured notes. This ensures the agent is evaluated against the reality of enterprise processes.

2.2 Stage 2: Structured Agent Definition

We convert source documents into a rigorous **Agent Definition**. This serves as the "blueprint" for the agent's expected behavior.

- **Metadata:** Versioning, domain tags, and complexity classification.
- **Core Logic:**
  - `role` & `goal`: High-level directive.
  - `action_plan`: A descriptive guide on how the agent should behave.
  - `workflows`: Specific sequences including `trigger_conditions`, `steps`, and `exit_conditions`.
- **Tools:** Distinct lists for `system_tools` (standard utilities) and `domain_tools` (business-specific APIs and Automations).
- **I/O Schema:** Strictly typed `input_variables` and `output_variables` (defining name, type, required/optional status) to ensure the agent adheres to enterprise data contracts.

2.3 Stage 3: High-Level Scenarios & Milestones

From the definition, we generate **High-Level Scenarios**. This layer bridges the gap between abstract goals and concrete testing.

- **Representative Inputs:** Instead of generic placeholders, we generate specific business data (e.g., `order_id`, `sku`, `required_quantity`).
- **Rationale & Context:** The `rationale` field explains *why* a specific test case exists (e.g., "Testing error recovery when SKU is out of stock").
- **Milestones:** An array of `milestones`. Each milestone contains a `step`, `description`, and `expected_state`. This allows our Trajectory Evaluator to grade the path the agent took, not just the final output.

- **Success Criteria:** Defined `expected_success_outcome` and `success_path_differences` to handle edge cases where multiple valid paths exist.

2.4 Stage 4: Executable Test Classes

The final stage converts scenarios into **Executable Python Test Classes**.

- **Code Generation:** The pipeline generates valid Python code where each test case inherits from a `BaseAgentNameTestCase`.
- **Runtime Injection:** `input_data` is injected directly into the agent's context.
- **Assertion Logic:** The class includes `expected_tool_calls` and validation logic to programmatically verify if the agent hit the required milestones during execution.

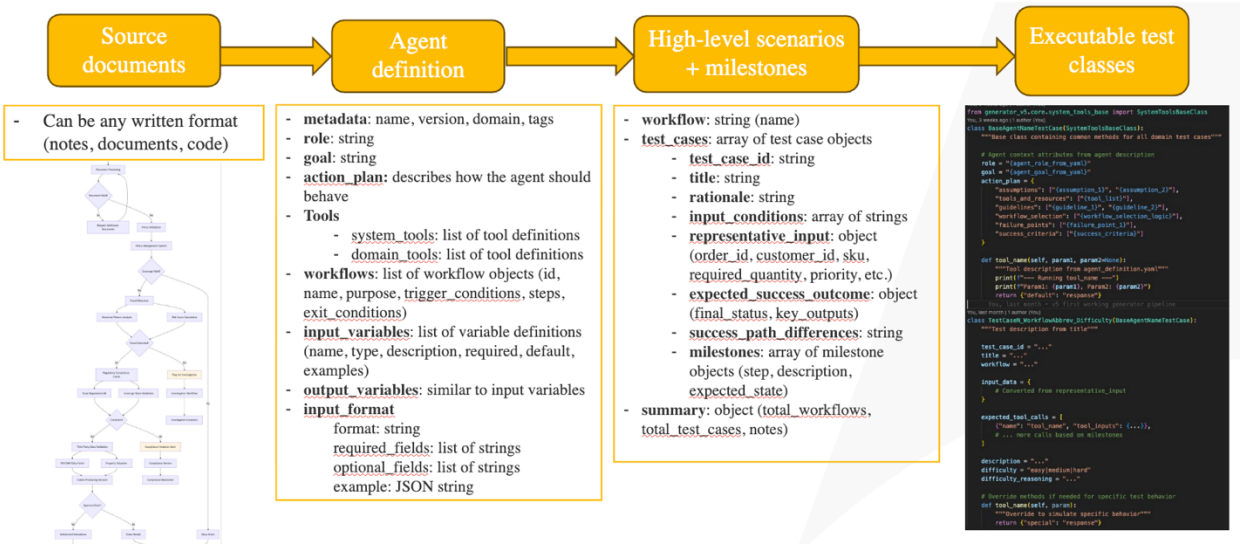


Figure 2.1: The APA-Eval V5 Generation Pipeline, moving from unstructured source data to executable Python test classes.

3. Evaluation Framework

To capture the nuance between a "lucky guess" and a "reliable agent," we employ a dual-metric framework. This approach ensures we measure both the destination (Task Success) and the journey (Trajectory Accuracy).

3.1 Task Success Evaluator

Purpose: Evaluates strictly whether the AI agent completed both successfully and accurately the intended business objective.

Criteria:

- **Completeness**: Does the response address all parts of the user's request?
- **Tool Usage**: Were tools used correctly without omitting necessary parameters?
- **Relevance**: Is the final output free of extraneous or distracting information?

Validation: To ensure this metric aligns with industry standards, we benchmarked it against **Tau2-bench**, a leading tool-use dataset. Our Task Success Evaluator achieved a top score on Tau2-bench, confirming its reliability in identifying correct outcomes.

3.2 Trajectory Accuracy Evaluator

Purpose: Assesses the logical quality of the agent's execution path, independent of the final result. This is our primary defense against "Scrappy Wins" (where the agent goal is achieved, but through inefficient or risky execution paths).

Criteria:

- **Logical Sequencing**: Did the agent perform steps in the correct dependency order?
- **Efficiency**: Were there unnecessary tool calls or redundant loops?
- **Grounding**: Did the agent avoid hallucinations or unfounded assumptions during the reasoning process?

Industry Comparison: We compared our internal **Trajectory Accuracy** metric against Galileo's "Action Advancement" metric, a leading commercial standard for agent evaluation.

On our in-house dataset, our Trajectory Accuracy Evaluator demonstrated superior alignment:

- **Our Metric: 0.86**
- **Galileo Action Advancement: 0.73**

This **17% improvement** indicates that generic evaluation metrics often fail to capture the specific constraints of business process automation.

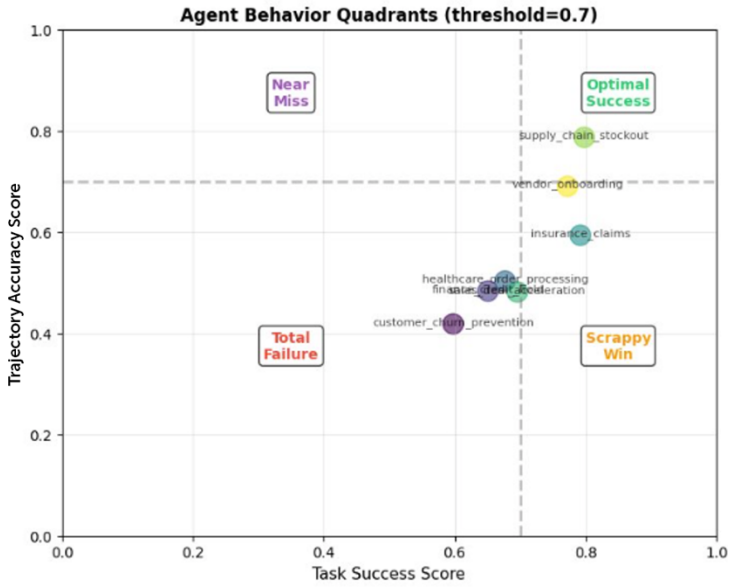
4. Empirical Results

We evaluated a diverse suite of Large Language Models (LLMs) against our evaluation dataset. The results validate our hypothesis: high task success rates can mask underlying reasoning failures.

4.1 The Quadrant Analysis: Classifying Agent Behavior

By plotting **Task Success** (x-axis) against **Trajectory Accuracy** (y-axis), we identified four distinct behavioral patterns in autonomous agents.

1) **Optimal Success (Top-Right):** The agent achieves the goal using the intended, efficient workflow. This is the target state for production.
2) **Scrappy Win (Bottom-Right):** The agent achieves the correct final outcome but follows a suboptimal, risky, or inefficient path.
    a. *Risk:* These agents appear successful on standard benchmarks but carry hidden technical debt (e.g., guessing parameters, skipping safety checks).
3) **Near Miss (Top-Left):** The agent follows the correct procedure perfectly but fails on the final output (often due to minor formatting or omission errors).
4) **Total Failure (Bottom-Left):** The agent fails both the reasoning process and the final objective.



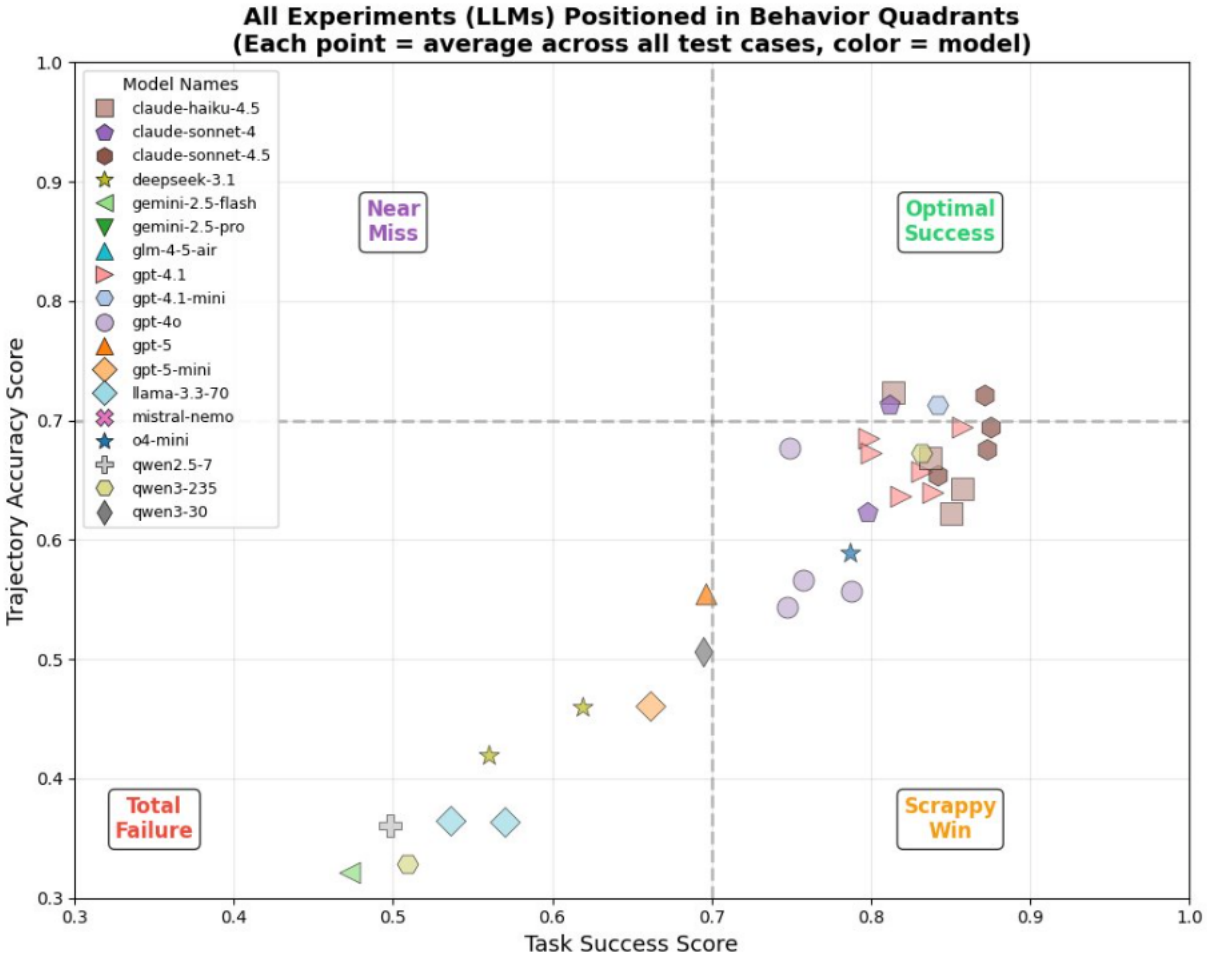


Figure 4.1: Agent Behavior Quadrants. Note the cluster of results in the "Scrappy Win" region, indicating agents that succeed superficially but fail procedurally.

4.2 The "Scrappy Win" Phenomenon

Our data reveals a critical insight: **Suboptimal runs do not distribute randomly**. They cluster heavily in the "Scrappy Win" quadrant.

As shown in our divergence analysis, specific agent types—particularly `sales_deal_acceleration` and `customer_churn_prevention` showed a **Divergence Rate (Delta > 0.3)** of over 40% of total cases. This means that for complex, multi-step tasks, agents are frequently "lucking into" the right answer without understanding the underlying business logic.

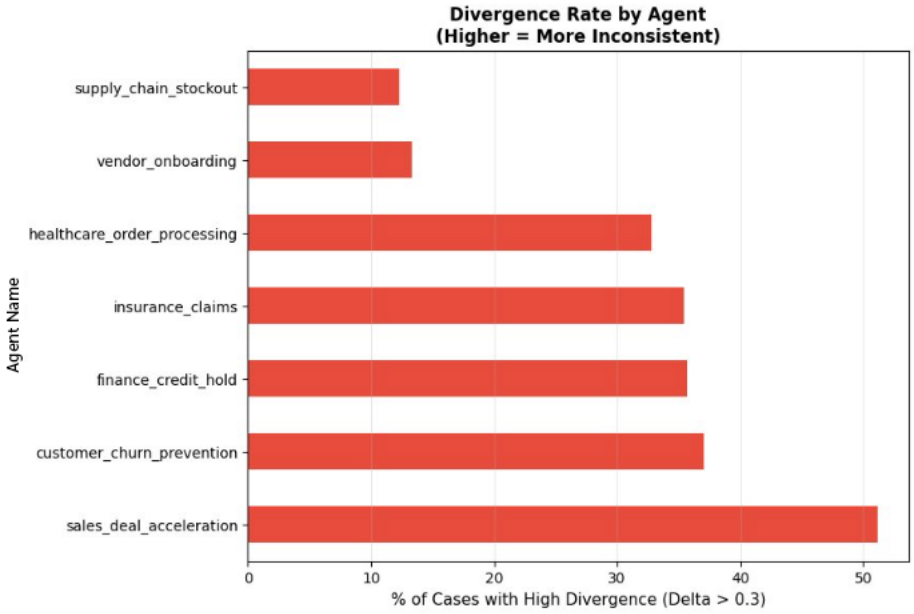


Figure 4.2: Divergence Rate by Agent. The red bars indicate the percentage of cases where Task Success significantly exceeded Trajectory Accuracy.

4.3 LLM Performance Rankings

We benchmarked leading proprietary and open-source models to assess their readiness for agentic workflows.

Note: Results represent a point-in-time snapshot and may not reflect the latest model rankings.

Key Findings:

1) **The Leaders: Claude 4.5 Sonnet** and **GPT-4.1** consistently inhabit the "Optimal Success" quadrant, showing high correlation between their reasoning and their results.
2) **The "Scrappy" Cohort:** Smaller or older models (e.g., **GPT-4o-mini, Llama-3.3-70**) often show respectable Task Success scores but suffer from significantly lower Trajectory scores. For example, while `gpt-4o-mini` achieves a high task success rate, its behavior classification reveals a higher incidence of "Scrappy Wins" (22.0%) compared to `gpt4.1` (13.6%).

Behavior Categories by LLM (Experiment)

behavior_category	Optimal Success	Scrappy Win	Near Miss	Total Failure
gpt-4.1	64.4	13.6	5.1	16.9
claude-haiku-4.5	62.7	18.6	10.2	8.5
claude-sonnet-4.5	61.0	25.4	5.1	8.5
gpt-4.1-mini	61.0	22.0	10.2	6.8
qwen3-235	54.2	27.1	5.1	13.6
claude-sonnet-4	52.5	23.7	15.3	8.5
gpt-4o	50.8	18.6	8.5	22.0
o4-mini	42.4	28.8	5.1	23.7
gpt-5	37.3	25.4	10.2	27.1
qwen3-30	30.5	33.9	6.8	28.8
deepseek-3.1	28.8	20.3	6.8	44.1
gpt-5-mini	23.7	32.2	8.5	35.6
qwen2.5-7	15.3	22.0	5.1	57.6
llama-3.3-70	13.6	23.7	6.8	55.9
gemini-2.5-flash	11.9	18.6	6.8	62.7
gemini-2.5-pro	6.8	13.6	1.7	78.0
glm-4-5-air	0.0	8.5	0.0	91.5
mistral-nemo	0.0	3.4	1.7	94.9

Overall Performance by LLM

	task_success	trajectory	milestone	avg_all_metrics
claude-sonnet-4.5	0.87	0.72	0.77	0.79
claude-haiku-4.5	0.81	0.72	0.79	0.77
gpt-4.1	0.86	0.69	0.75	0.77
gpt-4.1-mini	0.84	0.71	0.75	0.77
qwen3-235	0.83	0.67	0.77	0.76
claude-sonnet-4	0.81	0.71	0.71	0.74
gpt-4o	0.75	0.68	0.71	0.71
o4-mini	0.79	0.59	0.72	0.70
gpt-5	0.70	0.56	0.69	0.65
qwen3-30	0.69	0.51	0.68	0.63
gpt-5-mini	0.66	0.46	0.71	0.61
deepseek-3.1	0.62	0.46	0.67	0.58
llama-3.3-70	0.57	0.36	0.65	0.53
qwen2.5-7	0.50	0.36	0.55	0.47
gemini-2.5-flash	0.47	0.32	0.51	0.43
gemini-2.5-pro	0.32	0.24	0.46	0.34
mistral-nemo	0.14	0.19	0.24	0.19
glm-4-5-air	0.17	0.08	0.30	0.18

Figure 4.3: Behavior Categories and Overall Performance by LLM. Note how top-tier models maximize "Optimal Success" while mid-tier models rely more heavily on "Scrappy Wins."

5. Implementation Insights & Best Practices

Building a robust evaluation framework is an iterative engineering challenge, not a one-off task. Through the development of APA-Eval V5, we encountered several friction points that offer valuable lessons for enterprise AI teams.

5.1 The Data Generation Trap

Insight: Data generation is deceptively difficult. A common pitfall is attempting to scale test case volume before validating test case quality.

- **The "Cleanup Debt":** Scaling up generation too early simply multiplies the cleanup effort. We found that generating 100 high-quality, manually verified "golden" examples is infinitely more valuable than 1,000 noisy, synthetic cases.
- **Multi-Pass vs. One-Shot:** We abandoned one-shot prompting for test generation. A multi-pass pipeline—where one step defines the agent, the next generates the scenario/milestones, and third re-writes those into code consistently yields higher fidelity test cases.
- **Code as the Universal Language:** We leverage the fact that modern LLMs excel at understanding and generating code. By structuring our test cases as executable Python classes rather than abstract text descriptions, we reduce ambiguity and enable automated execution.

5.2 Addressing Model Limitations

Insight: Even state-of-the-art models exhibit persistent blind spots, particularly regarding calculation and strict formatting.

- **The Math Gap**: LLMs continue to struggle with reliable arithmetic. In our experiments, agents frequently failed finance and inventory tasks not because of poor reasoning, but because of calculation errors.
    - *Recommendation*: Do not rely on the LLM for math. Integrate deterministic tools (e.g., Python script execution, calculator APIs) and force the agent to use them for any numerical operation.
- **Model Incompatibility**: Not all models are compatible with complex agentic evaluation. Some smaller or less aligned models consistently fail to adhere to the strict JSON schemas required for our tool definitions, rendering them untestable in a structured environment.

5.3 The Importance of Domain Fit

Insight: Generic benchmarks do not predict production success.

- **The "Use-Case" Rule:** The entire purpose of evaluation is to predict how an agent will behave in *your* specific environment. If the benchmark doesn't mirror your specific data structures, and business rules, the score is a vanity metric.
- **The Coverage Challenge:** Defining "100% coverage" for an autonomous agent is nearly impossible due to the infinite combination of edge cases. Instead of chasing total coverage, we focus on **iterative versioning**—each new version of the dataset is explicitly designed to test failure modes exposed by the previous version.

5.4 Monitoring the "Scrappy Win"

Insight: As noted in our Quadrant Analysis, agents often appear confident while drifting logically.

- **Production Monitoring:** In production, you cannot easily measure "Trajectory Accuracy" because you don't have a ground truth label. However, you *can* monitor for the characteristics of a "Scrappy Win," such as excessive tool calls, circular logic loops, or high uncertainty scores, even when the final result seems correct.

6. Future Directions

Our evaluation framework is not a static artifact but an evolving standard. Based on our current findings, we are expanding our research into three critical areas to further close the gap between offline benchmarking and production reality.

6.1 Advanced Agent Patterns & Architectures

Current benchmarks largely test "Action-Response" loops. We are expanding the dataset to evaluate more sophisticated cognitive architectures:

- **Sequential Thinking & Reflection**: Testing agents that "think before they act," explicitly planning steps and self-correcting before executing tool calls.
- **Multi-Agent Collaboration**: Evaluating scenarios where a primary agent must delegate sub-tasks to specialized worker agents, measuring the fidelity of information hand-offs.

6.2 Long-Term Memory

Real-world agents do not operate in a vacuum; they learn from user history. We are currently testing how scores evolve with including more context from our **Process Reasoning Engine (PRE)**. This involves evaluating adding even more targeted context from previous runs and automations.

6.3 The Self-Improving Pipeline

Our long-term vision is to close the loop between agent performance and dataset generation.

- **Adversarial Edge Cases:** We are expanding our generator to specifically target "Scrappy Win" scenarios—injecting ambiguity and error states to force agents to fail or demonstrate robust recovery.
- **Automated Expansion**: Build a feedback loop where the specific failure modes of poorly performing agents (e.g., customer_churn_prevention) automatically trigger the generation of new, harder test cases targeting those specific weaknesses.

7. Conclusion

As Business Processes transition to Enterprise Agents, the definition of success has shifted. It is no longer sufficient for a model to generate a plausible-sounding answer; it must execute a business process reliably, efficiently, and logically.

Through the development of the **APA-Eval V5 framework**, we have established that:

1. **Process Matters:** A "Scrappy Win", where an agent guesses its way to success, is a hidden liability. Measuring **Trajectory Accuracy** is the only way to detect these risks before deployment.
2. **Data is the Differentiator:** Generic benchmarks fail to capture enterprise complexity. Our purpose-built, code-based dataset generation pipeline allows us to test agents against the specific schemas and constraints they face in the real world.
3. **We Are Setting the Standard:** With a **93% accuracy on Tau-bench** (surpassing the public leaderboard) and Trajectory Accuracy Evaluator that is **17% better** than leading commercial tools, our framework provides a rigorous foundation for building trusted autonomous systems.

The agents that will define the future of enterprise automation are not just those that get the job done, they are those that get the job done ***right***. This framework ensures we can tell the difference.

AUTOMATION ANYWHERE