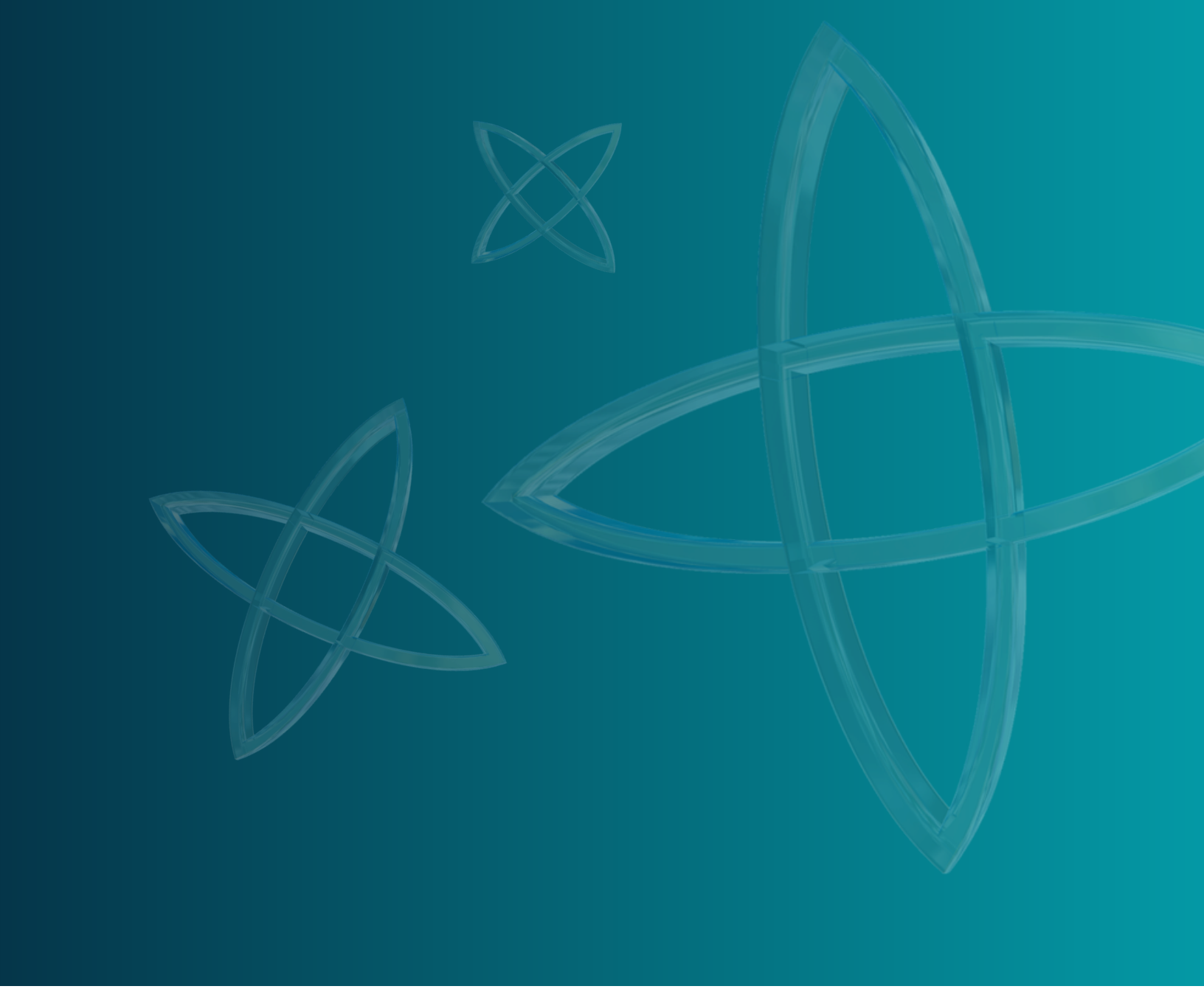

April 2026

Eval-Driven Development for Multi-Agent AI Systems: A Practitioner's Framework

How a small non-profit team uses composable evaluations to make model selection, cost optimization, and quality assurance measurable, not speculative.

Francis Beeson and Pam Portman

SKILLUP

Framing the Problem

Most AI products rely on anecdotal quality assessment. Someone tries the system, decides it "feels good" or "feels off," and adjustments are made based on intuition. For a prototype, this is fine. For a system that serves vulnerable workers navigating career crossroads, emotional distress, or complex life constraints, it is not.

SkillUp's AI Coach (SUAI) is a conversational AI career coach built for non-degree workers. It coordinates multiple specialized AI agents to deliver empathetic, personalized career guidance through voice and text, with persistent memory of each user across sessions and life changes. The system makes consequential decisions on every user message: Is this person in crisis? Should the conversation shift from career exploration to job search? Is the user's emotional state shifting from engagement to frustration? Are we respecting the constraints they told us about: childcare schedule, transport limitations, budget?

These decisions are made by AI agents, and each one must be verifiable. As AI systems move beyond single-model chatbots to architectures where multiple specialized agents coordinate to serve a user, the evaluation challenge changes fundamentally: it is no longer enough to test one model's output in isolation; you need to verify that the agents, working together across turns, produce the right outcome. When the stakes are this high, "it seems to work" is not an engineering standard. This paper describes the evaluation framework we built to replace intuition with evidence: what it measures, how it works, and what we learned building it - as we prepare to scale this to thousands of SkillUp users targeting later this year.

The Core Idea: Evals as Operational Infrastructure

Every AI system we build starts from the same discipline: define what "correct" looks like before writing the first line of agent logic. Build the evaluation framework first. Make it the inner loop of development.

Change a prompt, run evals. Swap a model, run evals. Adjust retrieval parameters, run evals. Every iteration is measured, every regression is caught, and every improvement is quantified.

This is not a testing phase at the end of a project. It is the development methodology itself. The evaluation framework shapes how agents are designed, how models are selected, how prompts are refined, and how the system is deployed. It produces three things that anecdotal quality assessment never can:

- Confidence in what you are shipping. Every release is backed by quantified evaluation results, not someone's opinion that the output "looks fine."

- Data for every decision. Model selection, prompt refinement, and architecture modifications are all evidenced. When a stakeholder asks, "why this model?", the answer is a comparison report, not a preference.

- Protection against silent regression. Model providers update their hosted models, sometimes in ways that subtly alter behavior without public announcement. New agents are added, prompts evolve, retrieval parameters shift. In any of these cases, the eval suite catches degradation before it reaches users. This is the AI equivalent of a safety net: the system's behavior is continuously validated against defined standards.

Three Layers of Evaluation

We architect evaluations around a three-layer model that moves from individual component validation to system-level outcome measurement. The analogy to software testing is deliberate.

Operational Evals (Unit Tests for Agent Nodes)

Operational evaluations target individual agent nodes: the atomic decision-making units of the system. Each agent receives an input and produces a structured output. Each eval case defines one scenario: this input must produce this output, validated against these rules.

A user says: "I just got evicted and I don't know what to do."

The evaluation framework validates that the Mode Selector (the agent that determines which conversational journey and attitude to adopt) correctly identifies this as a crisis situation and switches the system to crisis support mode. It checks that the attitude mode shifts to crisis support, because survival must come before career development. It verifies that the system does not continue with routine career coaching questions when a person is in acute distress.

This is not a matter of opinion. It is a testable, verifiable claim: given this input, the agent must produce this output. Either the Mode Selector routes this correctly, or it does not. The eval catches it either way.

Other operational evals in SUAI verify that the Gate Tracker (which monitors onboarding milestones like whether the user has shared their name, location, and basic background) accurately assesses which checkpoints have been completed. That the Observation Extraction agent correctly identifies "two kids under five" as a childcare constraint. The Memory Compaction agent consolidates fragmented employment history observations into a coherent career record without losing information.

Operational evals are fast, cheap, and provide precise, actionable feedback. Every agent ships with operational evals that define its expected behavior before it touches a single user.

Tactical Evals (Integration Tests for Workflows)

Tactical evaluations assess the quality of multi-agent workflows and ongoing conversations. Where operational evals test a single agent in isolation, tactical evals test how agents work together across multiple turns to serve a user.

Consider a practical failure that only tactical evaluation can catch: a user states early in a conversation that they have childcare constraints preventing night work. Fifteen turns later, the system recommends a night-shift position. Each individual agent in the chain did its job correctly. The intent was classified right. The job matching worked. The recommendation format was correct. But the system, as a whole, violated a hard constraint. A single violation, even in an otherwise excellent conversation, represents a failure of trust.

Tactical evals assess the emergent properties that arise from orchestrating multiple agents across a full conversation: Is the system respecting the user's stated preferences and constraints throughout the session and across sessions? Is the system managing information load appropriately, or is it overwhelming a first-time user with too many options? Is there coherence and consistency across turns, or is context being lost? Are there points where the conversation stalls, loops, or becomes inefficient?

These are properties that no amount of operational testing can reveal, because they only emerge from the interaction between agents across time.

Strategic Evals (End-to-End Tests for Outcomes)

Strategic evaluations measure system-level outcomes: the indicators that tell you whether the product is achieving its purpose.

Is the user making measurable progress toward their stated career goal across sessions? Did the system help them move from "I don't know what I want to do" to a concrete action plan with specific next steps?

Is the user's emotional trajectory improving over the course of the coaching relationship? Does a user who arrived in distress move over time toward confidence, agency, and engagement? For a system serving vulnerable populations, this is not a nice-to-have; it is a measure of whether the AI is doing no harm.

How much effort does the user have to invest to achieve their goal? A good AI coach reduces friction: it remembers what the user has already said, proactively surfaces relevant information, and handles administrative tasks so the user can focus on decisions. A User Effort Score measures the inverse of friction, and tracks it over time.

Strategic evals connect AI quality to human impact. They are what let stakeholders say, with evidence, "the system is working" or "the system needs attention in this area." Without them, the best you can offer is "all the unit tests pass," which, as any experienced engineer knows, is necessary but nowhere near sufficient.

Online and Offline Evaluation

SUAI currently runs all evaluations offline: during development (whenever a prompt is edited or a different LLM is tested) and during CI/CD (automated quality gates before deployment). This is deliberate. Online evaluations add latency, and SUAI maintains a strict 2-second end-to-end response time SLO (p50) for the conversational experience.

But the relationship between online and offline evaluation is more nuanced than "offline for development, online for production." It depends on what type of evaluator you are running and what it produces.

Operational evals cannot run online in most cases. An operational eval validates an agent's output against an expected output: given this input, the agent must produce this specific result. In production, you do not have the expected output; that is the whole point of running the agent. There are edge cases (a pre-mapped lookup of known inputs to expected outputs), but these tend to be situations where you may not need an LLM at all. Operational evals belong in development and CI/CD, where expected outputs are defined.

Scorers can run online, and this is where it gets interesting. Strategic evaluators like NPS, User Effort Score, or Goal Achievement Score apply quantitative metrics to qualitative cognitive judgments. They do not compare against an expected output; they assess a quality dimension and produce a score. This means they can run during a live conversation.

Consider a practical example: an NPS scorer that assesses, after each turn, whether the conversation is trending toward a score below a critical threshold. If the scorer detects a downward trend, procedural code can react in real time: injecting a prompt component that warns the narrator agent that the user may be having a negative experience, nudging the system to course-correct before the user disengages. The user never sees the evaluation; they experience a system that adapts.

Online and Offline Evaluation (cont.)

Or consider a tactical scorer that measures the information processing load the user is experiencing. If the narrator is providing too much information and the scorer signals that the user may be suffering from information overload, the system can react: simplify, slow down, proactively ask the user if a specific point is unclear. This is evaluation results flowing back into the orchestration pipeline to steer agent behavior, transforming evals from a passive measurement tool into an active quality enforcement mechanism.

The critical distinction is between evaluators that compare against expected outputs (which require offline definition) and scorers that assess quality dimensions against criteria (which can operate on live data). Both are valuable. The architecture needs to support both, and the engineering team needs to be deliberate about which runs where, because every online evaluator adds latency overhead, and in a system with tight response time constraints, that overhead is not free.

Three Patterns That Make It Work

Composable Validation

An agent's output is not a single value. It is a structured object with multiple fields, and each field has different validation requirements. SUAI's Mode Selector, for instance, returns a journey mode (an enum classification that must be exactly right), an attitude mode (another enum that might accept multiple valid values for an ambiguous scenario), and a justification (a free-text string where phrasing will vary but key reasoning must be present). A single equality check across the entire output object is too brittle: the justification will never match word-for-word, so the test fails even when the classifications are correct. But checking only the classifications ignores whether the agent's reasoning is sound. What you need is the ability to validate each field with a different strategy, composed together into a single eval case.

Add to this that LLM outputs are inherently variable. The same correct answer can be phrased differently every time. A loose "looks about right" check misses real regressions. A strict equality check catches too many false failures.

The solution is composable field validators, each one a precision tool for a specific validation need:

Exact requires a perfect match. In SUAI's Mode Selector evals, both the journey mode and attitude mode are validated with Exact: the journey mode must be exactly CRISIS_SUPPORT for a crisis scenario. There is no room for ambiguity in a crisis detection decision.

OneOf allows a field to be any one of a set of acceptable values. If a user's message is emotionally ambiguous, an eval case could accept the attitude mode being either EMPATHETIC_LISTENING or COACH, since both would be reasonable responses to the context.

Substring checks that a text field contains a particular string somewhere within it, useful when exact phrasing varies but a key concept must be present. For example, validating that a memory compaction summary mentions "childcare" without requiring an exact sentence.

Composable Validation (cont.)

Contains verifies that a list field includes at least a specified set of values. In SUAI's memory compaction evals, this validates that the list of observation IDs feeding an operation includes the specific observations the agent should have used.

AllOf ensures a list field contains every item in a required set, for scenarios where all specified elements must be present.

ListMatches validates complex structured lists by checking that the list contains items matching a specification, where each item can use different validators for different sub-fields. In SUAI's skills compaction evals, this validates that the agent's output operations list contains an entry where the field is exactly SKILLS, the action is exactly APPEND, and the summary meets a semantic criterion, all within the same list item. Extra items are permitted: the eval checks "at least these," not "exactly these."

Criteria invokes an LLM as a judge to evaluate a field against human-readable criteria, used where rule-based validators are insufficient. In SUAI's compaction evals, the summary field of each memory operation is validated with criteria such as "the summary should reference the specific skill mentioned by the user and explain why it was added." Criteria validation is applied per field, not per case, so semantic evaluation is targeted rather than blanket.

These validators compose freely within a single eval case. The result is scalpel-like precision: strict where strictness matters (crisis detection must be exact), flexible where variation is acceptable (a compaction summary's phrasing can vary as long as the key information is present). A three-tier priority governs validation within each case: field-level validators take precedence, falling back to full output equality if none are specified, and failing explicitly if no validation is defined. No silent defaults.

Adapter Injection for Model Comparison

The AI landscape moves fast. New models release frequently, pricing changes, and performance characteristics shift. Rather than relying on vendor benchmarks or anecdotal impressions, the evaluation framework runs the exact same eval suite against multiple LLM providers simultaneously, producing structured comparison reports.

An adapter resolver maps model identifiers to the correct client across providers (Anthropic, Google Vertex AI, AWS Bedrock, Groq, and others), so the same eval case runs identically regardless of provider. A single command compares three models from different providers on the same agent with identical inputs. The output is a structured report showing, per model: pass rate, latency distribution (average, P50, P95, P99), cost per call, and a per-case breakdown showing where specific models excel or struggle. Multiple runs per model provide statistical stability: distributions and variance, not single data points.

This answers the question that matters: which model should this specific agent use in production, given its accuracy requirements, latency budget, and cost constraints? Different agents in the same system can, and should, use different models. A safety-critical agent like crisis detection needs the most accurate model available. A background summarization agent can use a cheaper, faster model if it maintains acceptable accuracy. Model selection becomes a data-driven decision, not a vendor preference.

Evaluation Ownership: From Engineers to Domain Experts

Eval case definitions are declarative: they specify an input scenario, the expected output, and validation rules. Defining an eval case does not require understanding the agent's internal implementation; it requires understanding what correct behavior should be for a given situation.

This is a deliberate architectural separation. The people closest to the workers define what correct behavior looks like; engineers build and maintain the framework machinery that enforces it.

A product manager who has spent years working with the population SUAI serves knows what "appropriate crisis response" means in practice. A program director at a partner organization knows what constraints their users face and what a respectful, effective coaching interaction looks like. These are the people who should be defining evaluation cases, because they hold the domain knowledge that determines quality.

The current friction is that case definitions live in code. The content is readable, but the format is unfamiliar to non-developers. The planned evolution is to capture evaluation cases through accessible interfaces, whether structured spreadsheets, web-based forms, or graphical tools, and have the application map them to the framework's internal format. This removes the last barrier to non-developer authorship while preserving the full expressiveness of the validation system.

The implication for scaling is direct: evaluation coverage grows as domain expertise grows, not as engineering headcount grows. Every new partner, every new program context, every new population brings domain knowledge that translates into evaluation cases. The framework becomes richer as the ecosystem expands, without requiring proportional growth in the engineering team. **The people who understand the population define what quality means; the framework enforces it at scale.**

What We've Learned

Cost, latency, and accuracy are inseparable

Every evaluation run tracks three dimensions for every agent call: accuracy (did it produce the correct outcome?), latency (how fast?), and cost (how many tokens, how many dollars?). These three dimensions measured together are what transform model selection from a technical preference into a business decision.

A model might be highly accurate but too slow for a conversational interface. Another might be fast and cheap but not accurate enough for a safety-critical decision. Without measuring all three simultaneously, on the same eval cases, you are making trade-offs you cannot see.

For example, SUAI's memory compaction agents, which process user profile data in the background, currently run on a premium model. The evaluation framework is what enables safe migration to a more cost-effective alternative: by running the full eval suite on both models, the team sees precisely whether accuracy is maintained, how latency changes, and what the cost savings would be at scale. This is the kind of evidence-based cost management that enables responsible scaling, and it is only possible because cost is a first-class dimension of every evaluation, not an afterthought.

What We've Learned

Evaluation schemas and production schemas are not the same thing

Consider the Mode Selector output described earlier: journey mode, attitude mode, and a justification string. In production, the justification field is not consumed by any downstream agent or user-facing component. Nobody reads it at runtime. But every output token the LLM generates costs latency and money. So in production, the output schema omits the justification field entirely. The agent returns only the classifications it needs to return.

During development and QA, the schema includes the justification. This is where the Criteria validator earns its value: it evaluates whether the agent's reasoning is sound, not just whether the final classification is correct. If evaluation results start showing drift or unexpected failures, the reasoning field is what helps the engineering team understand why the model is making a particular decision, not just that it got it wrong.

The evaluation framework must accommodate this. The output model you test against during offline evaluation may include fields that you would never generate in production, because those fields exist specifically to make the agent's reasoning inspectable and evaluatable. This is a small architectural detail, but it reflects a broader principle: evaluation and production have different requirements, and the framework should not force them into the same shape.

What We've Learned

Response time SLOs enforce architectural discipline

SUAI maintains a strict 2-second end-to-end response time SLO (p50). The conversational experience, especially in voice mode, must feel natural. Long pauses break conversational flow and make the interaction feel robotic. To meet this target, the entire multi-agent orchestration workflow must complete in under one second, leaving the remaining second for the narrator agent to begin generating its conversational response.

This constraint has profound architectural implications. It means SUAI cannot use autonomous agents (what some call "LLMs with tools in a loop") for its core orchestration. Autonomous agents, where the LLM decides at runtime what to do and how to do it through dynamic task decomposition, are powerful when the problem genuinely cannot be decomposed in advance: research tasks, open-ended exploration, novel problem-solving. But they are slow. They involve multiple LLM calls, tool invocations, reasoning loops, and retries. They cannot reliably meet a sub-second orchestration budget.

Most enterprise AI problems do not actually require dynamic task decomposition. They are automating processes for which a standard operating procedure already exists, explicitly or implicitly. There is a "way to do this" inside every organization. What these processes need is static task decomposition: a designed orchestration of specialized agent nodes, each handling a narrow cognitive task with defined inputs and outputs, following a workflow that mirrors how humans in the organization already handle the process. Each agent has degrees of freedom for the cognitive judgment it performs, but the data flow, the decision points, and the handovers between agents are architecturally constrained.

The SLO forces this discipline. When you must respond in under two seconds, you cannot afford a reasoning loop that takes fifteen seconds. You cannot afford online evaluations that trigger retries if an agent gets something wrong. You cannot afford the token burn of an autonomous agent calling an LLM six times when a well-designed orchestration calls it once per node. This is also why multi-model comparison matters so much in practice: some providers offer significantly lower latency for comparable accuracy. SUAI uses models hosted on Groq (which uses specialized LPU hardware optimised for inference throughput) for orchestration subagents where latency is the binding constraint. The evaluation framework is what makes this selection rigorous: same eval cases, same accuracy requirements, different providers, measurable trade-offs.

What We've Learned

Design the cognitive architecture, or pay for not doing it

Demanding a response time SLO, even when the business does not initially specify one, is one of the most productive things an engineering team can do. It forces you to design the cognitive architecture of the system deliberately: sit down with domain experts, understand the process being automated, and map every cognition, every decision, every data flow, every handover, every transformation. What are the inputs? What are the outputs? Where can agents run in parallel? Where must they run sequentially? Where does an LLM add value, and where is procedural code sufficient? Which decisions carry low responsibility if they fail, and which carry liability that demands human judgment?

This is the hardest and most valuable work in AI engineering. It is also the work that much of the industry is skipping.

The prevailing approach is to hand the entire problem to a powerful LLM, give it tools, and let it figure things out at runtime. This is not engineering. It is architectural laziness. And it is producing exactly the outcomes the industry is now grappling with: AI projects that cost orders of magnitude more in token spend than they need to, that respond too slowly for real-time use cases, and that cannot achieve the accuracy enterprise processes demand. Eighty percent accuracy sounds impressive until you realize it means one in five users gets an error or irrelevant output. In enterprise, that is not a minor quality issue; it is a product that does not work. It is also a product that will not deliver return on investment, because the token costs of an unconstrained autonomous agent burning through reasoning loops are orders of magnitude higher than a designed orchestration that calls each model once per node.

The solution is not hoping that the next model release will close the accuracy gap. It is doing the engineering work: sitting down with the people who understand the process, mapping the cognitive architecture, designing the orchestration, constraining each agent to the specific task it needs to perform, and then using the evaluation framework to verify that each agent and each workflow meets the accuracy, latency, and cost requirements the product demands. SLOs make this work unavoidable. Without them, the path of least resistance will always be to throw a more powerful model at the problem and hope for the best.

What We've Learned

Responsibility determines the automation boundary

Once the cognitive architecture is mapped, there is one more dimension that must be assessed before deciding what to automate: the responsibility each decision carries if it goes wrong.

Some decisions are low-stakes. If a career coach recommends a program that is not quite the right fit, the user can dismiss it and move on. The cost of failure is a minor friction in the conversation. Other decisions are not like this at all. In domains like legal document processing, financial compliance, or medical triage, a single incorrect decision at a specific point in the workflow can carry liability measured in millions of dollars, regulatory penalties, or direct harm to a person.

The evaluation framework is what makes this assessment evidence-based rather than speculative. When you can measure the accuracy of each agent node individually, you can compare that accuracy against the responsibility the node carries. A node that achieves 99.8% accuracy on a low-responsibility task is comfortably automated. The same 99.8% accuracy on a task where the 0.2% failure case could be catastrophic may not be. That node may need to be a human handover point: the orchestration pauses, a human makes the decision, and the workflow resumes with the human's input feeding the next agent in the chain.

This is not a theoretical concern. It is a design decision that should be made explicitly for every node in the cognitive architecture, and it can only be made with data. Without an evaluation framework that measures per-node accuracy, the team is guessing which tasks are safe to automate. With it, the boundary between human and machine judgment is drawn with evidence, and that boundary can be revisited as models improve, as eval cases expand, and as the team gains confidence in specific nodes over time.
For SUAI, crisis detection is the clearest example. When a user signals acute distress, the system's response is not a matter of convenience; it is a matter of safety. The evaluation framework must demonstrate, with evidence, that the agent handling this decision meets the accuracy standard the responsibility demands. If it cannot, the design must include a human escalation path at that point in the workflow.

What We've Learned

Domain experts are the bottleneck, not the framework

The hardest part of building good evals is not the technical infrastructure. It is getting the right person to define what "correct" looks like. The principal domain expert who truly understands the nuances of the use case is almost always scarce, busy, or both. Building the LLM judge is a matter of iterative prompt engineering. Getting the expert to articulate their judgment criteria clearly enough to encode is the actual constraint.

We have found that binary pass/fail judgments with detailed written critiques produce far better eval cases than Likert-scale ratings. When an expert says, "this is wrong because the system should have acknowledged the emotional state before redirecting to action," that critique becomes a Criteria validator. When an expert rates a response "3 out of 5," nobody knows what that means, including the expert next week.

The real measure of AI success

The AI industry's most visible successes right now are development tools: code generation, AI pair programmers, no-code builders. These are genuine achievements. But they measure a narrow kind of value: developer productivity. The output is more code, faster.

Outside of development tooling, the picture is less encouraging. Research consistently shows that most enterprise AI investments are not delivering returns. The projects struggling are not the ones helping developers write software faster; they are the ones trying to build AI products that serve real users with complex, consequential needs. This is where evaluation discipline, cognitive architecture design, response time constraints, and responsibility mapping matter most. These are the hard problems. They do not get solved by better code generation.

For the career development ecosystem, this distinction matters profoundly. The purpose of SUAI is not to ship features faster or generate more code. It is to help workers in crisis get the right response. To help someone navigate a career transition receive guidance that respects their constraints. To reach millions of people who need personalized career coaching and cannot access it through traditional channels. The only way to reach them at scale is through AI. And the only way to ensure that AI works for them is through rigorous, continuous evaluation.

What We've Learned

The real measure of AI success (cont.)

This is also a question of stewardship. Non-profit organizations building AI products operate with resources that funders have entrusted to them for a specific mission. The evaluation framework is what makes us responsible, efficient, and accountable with those resources. Responsible: every agent, every model choice, every architectural decision is validated against defined quality standards before it touches a worker. Efficient: per-agent cost tracking, multi-model comparison, and deliberate architecture design ensure that token spend is optimized, not wasted. Accountable: when a funder asks, "how do you know the system is working?", the answer is not an anecdote or a demo. It is evaluation data: pass rates, accuracy by agent, cost per interaction, latency distributions, and evidence of how quality is maintained as the system scales.

The measure of AI success, for us, is not how much code we can generate. It is whether workers are earning more, advancing in their careers, and reaching outcomes they could not have reached without support. These are outcomes that require sustained engagement with the worker and lasting partnership with the organisations that serve them. The evaluation framework is what ensures that the resources entrusted to us for that mission are being used with the rigour they deserve.

Acknowledgements

The research reported here was supported by the Gates Foundation, Google, the Cognizant Foundation, the GitLab Foundation, Truist Foundation, the Schultz Family Foundation, the Michael & Susan Dell Foundation, Charles Koch Foundation, and others through grants to SkillUp Coalition. The findings and conclusions in this report do not necessarily represent the official positions or policies of the funders.

We also thank the SkillUp users who participated in our research and allowed us to learn from their feedback and experiences.

SKILLUP

SKILLUP

www.skillup.org

Francis Beeson is the Principal AI Engineer at SkillUp Coalition and Director of Engineering at Mediux Coretx. Pam Portman is SVP of Product at SkillUp Coalition. They are working together to evolve SkillUp's AI-powered career navigation platform.

Francis Beeson
Principal AI Engineer at SkillUp Coalition & Director of Engineering at Medius Cortex

Pam Portman
SVP of Product at Skillup Coalition

Interested in working together? Connect at skillup.org/partners